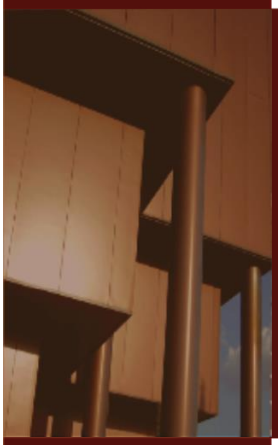
Trabajo Fin de Grado
Ingeniería Electrónica, Robótica y Mecatrónica

INTEGRACIÓN DE ROBOT SOCIAL PARA ASISTENCIA AL CLIENTE EN EL SECTOR DEL RETAIL

Autor: Adrián Cardona Ruiz

Tutor: Carlos Bordons Alba



Dep. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Trabajo Fin de Grado

Ingeniería Electrónica, Robótica y Mecatrónica

INTEGRACIÓN DE ROBOT SOCIAL PARA ASISTENCIA AL CLIENTE EN EL SECTOR DEL RETAIL

Autor:

Adrián Cardona Ruiz

Tutor:

Carlos Bordons Alba

Profesor titular

Dep. de Ingeniería de Sistemas y Automática

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2017

Trabajo Fin de Grado: INTEGRACIÓN DE ROBOT SOCIAL PARA ASISTENCIA AL CLIENTE EN EL SECTOR DEL RETAIL

Autor: Adrián Cardona Ruiz
Tutor: Carlos Bordons Alba

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2017

El Secretario del Tribunal

A mis padres

Agradecimientos

Quiero agradecer a los profesores Carlos Bordons Alba y Miguel Ángel Ridao Carlini la oportunidad que se me dio para colaborar en este proyecto. Gracias a este, he mejorado habilidades como futuro ingeniero, tales como pueden ser la programación en distintos lenguajes, y otras habilidades que son igual de útiles y de las que no se tienen tanta importancia, tales como la paciencia a la hora de resolver el problema propuesto, la búsqueda de información específica dentro de un gran catálogo como puede ser Internet.

Todo esto ha hecho que mi manera de trabajar, de pensar y de afrontar los problemas evolucione. Personalmente, esta evolución la encuentro más interesante que el aprendizaje de un lenguaje de programación, ya que son habilidades que son más difíciles de entrenar y perfeccionar.

Por otro lado, también quiero agradecer la ayuda prestada a Francisco Javier Buenavida Durán, Adrián Fernández Sola y Gonzalo Hernández Rodríguez que fueron los que me iniciaron en ROS e hicieron que mi aprendizaje fuera más dinámico e interesante.

Adrián Cardona Ruiz
Estudiante de Ingeniería Electrónica, Robótica y Mecatrónica
Sevilla, 2017

Resumen

Las empresas buscan crear una relación empresa-cliente única y por ello uno de los puntos en lo que se basan es en facilitar o mejorar la experiencia del cliente con ellos.

De ahí, la necesidad de este proyecto. Además, dicho trabajo es útil para evaluar hasta qué punto es conveniente el uso de un autómata como mediador. Todo esto está contenido dentro del proyecto RETAIL en el cual colaboré con AICIA.

Más adelante en este documento detallaré los framework que se usan, pero como punto a resultar será el uso de la herramienta ROS (Robotic Operating System), pero esta ya con un enfoque de aprendizaje, dado que en el proyecto colaboraban más personas que ya la conocían.

Por lo tanto, el objetivo de este trabajo fin de grado es doble. Por un lado, seguir formándome en cuanto a lenguajes y arquitecturas de comunicación entre máquinas y la creación de todos los programas para la comunicación del robot con el usuario.

Abstract

The Companies seek to create a unique company-customer relationship and for that reason one of the points in which they are based is in facilitating or improving the customer experience with them.

From there, the need for this project. In addition, such work is useful to evaluate until it is convenient point to use an automaton as mediator. All of this is contained within the RETAIL Project in which I collaborated with AICIA.

Later in this document I will detail the frameworks that are used, but as a result will be the use of the ROS (Robotic Operating System) tool, but this is already a learning approach, since the project collaborated more people who already knew.

Therefore, the aim of this end-of-degree paper is twofold. On the one hand, continue to form me as languages and communication architectures between machines and the creation of all programs for the communication of the robot with the user.

Índice

Agradecimientos **18**
Resumen **20**
Abstract **22**
Índice **24**
Índice de Figuras **26**
1. Introducción y Objetivos **29**
2. Descripción del robot Nao **31**
2.1 Lista de sensores y actuadores *31*
2.2 Versión y tipo de cuerpo *32*
2.3 Fuera de la caja *35*
2.4 Primera configuración *37*
2.4.1 Conectividad 37
2.4.2 Selección del idioma 39
2.4.3 Configuración de la conexión WiFi 39
2.4.4 Personalización del robot 41
2.4.5 Mi conectividad 43
3. Entornos de programación NAO **47**
3.1 Choregraphe *47*
3.2 Python y Python SDK *51*
3.3 C++ y C++ SDK *52*
4. Desarrollo del proyecto **53**
4.1 Evolución del proyecto *53*
4.2 Problemas *54*
4.3 Soluciones *54*
5. Integración del robot NAO en ROS **56**
5.1 Introducción a ROS *56*
5.2 Mensajes y Servicios *57*
5.3 Uso del robot real *57*
5.4 Uso del robot simulado *59*
5.5 Mi experiencia. *61*
6. Conclusión **66**
6.1 Resultado del Proyecto *66*
6.2 ¿Qué ha supuesto el TFG como persona/ingeniero? *68*
7. Anexos **69**
7.1 Manual de Usuario *69*
7.1.1 Instalación *69*
7.1.1.1 Instalación Choregraphe 69

7.1.1.2 Instalación PythonSDK 74
7.1.1.3 Instalación C++ 80
7.1.1.3.1 Windows 80
7.1.1.4 Instalación Jetty 83
7.1.1.5 Instalación ROS 85
7.1.2 Funcionamiento *89*
7.2 Programas *92*
7.2.1 Conocer version NAO 92
7.2.2 Código Nao 93
7.2.3 Programa exploración Turtlebot 96
7.2.4 Servidor NAO 100
7.2.5 Servidor simulado empresa 105
7.2.6 Navegación estanterias Turtlebot 107
7.2.7 Librería navegación estanterías 123
8. Referencias **126**

Índice de Figuras

ILUSTRACIÓN 1 ROBOT NAO. 29
ILUSTRACIÓN 2 LISTA DE ARTICULACIONES Y ACTUADORES. 31
ILUSTRACIÓN 3 VERSIÓN V5. 32
ILUSTRACIÓN 4 VERSIÓN V4. 33
ILUSTRACIÓN 5 VERSIÓN V3.3. 33
ILUSTRACIÓN 6 VERSIÓN V3+, V3.2. 33
ILUSTRACIÓN 7 TIPO DE CUERPO H25. 34
ILUSTRACIÓN 8 TIPO DE CUERPO H21. 34
ILUSTRACIÓN 9 TIPO DE CUERPO T14. 35
ILUSTRACIÓN 10 TIPO DE CUERPO T2. 35
ILUSTRACIÓN 11 POSICIÓN DE SEGURIDAD DEL NAO. 36
ILUSTRACIÓN 12 CONEXIÓN DE LA BATERÍA. 36
ILUSTRACIÓN 13 CONEXIÓN DEL ETHERNET. 36
ILUSTRACIÓN 14 FINALIZACIÓN DE MONTAJE. 37
ILUSTRACIÓN 15 PROCESO DE ENCENDIDO. 37
ILUSTRACIÓN 16 IP DEL ROBOT. 38
ILUSTRACIÓN 17 PROCESO DE ACCESO. 38
ILUSTRACIÓN 18 ASISTENTE DE CONFIGURACIÓN. 39
ILUSTRACIÓN 19 CONDICIONES DEL CONTRATO DE LICENCIA 39
ILUSTRACIÓN 20 EJEMPLO DE REDES 40
ILUSTRACIÓN 21 CONFIGURACIÓN DE LA RED DESEADA. 40
ILUSTRACIÓN 22 CAMBIO DE NOMBRE 41
ILUSTRACIÓN 23 CAMBIO DE CONTRASEÑA 41
ILUSTRACIÓN 24 ACCESO A ALDEBARAN CLOUD. 42
ILUSTRACIÓN 25 MENSAJE DE AVISO PARA VALIDAR LA CONFIGURACIÓN. 42
ILUSTRACIÓN 26 PROCESO FINALIZADO. 42
ILUSTRACIÓN 27 CREACIÓN DE LA RED HOSPEDADA EN WINDOWS. 43
ILUSTRACIÓN 28 PROCESO PARA COMPARTIR INTERNET A LA RED HOSPEDADA. 44
ILUSTRACIÓN 29 MENÚ DE CONFIGURACIÓN DE WINDOWS 10. 45
ILUSTRACIÓN 30 CREACIÓN DE LA RED HOSPEDADA DE FORMA SENCILLA. 45
ILUSTRACIÓN 31 PANEL SUPERIOR DE UBUNTU 14.04. 46
ILUSTRACIÓN 32 MENÚ DE CONFIGURACIÓN DE LA RED. 46
ILUSTRACIÓN 33 COMPROBACIÓN DE LA CREACIÓN DE LA RED. 47
ILUSTRACIÓN 34 CHOREGRAPHE. 48
ILUSTRACIÓN 35 DISTRIBUCIÓN DE LOS PANELES DE CHOREGRAPHE. 48
ILUSTRACIÓN 36 BREVE EJEMPLO DE CREACIÓN DE UN PROGRAMA. 49
ILUSTRACIÓN 37 CONTENIDO DEL INTERIOR DE LA CAJAS. 50
ILUSTRACIÓN 38 ROBOT STAIR. 56
ILUSTRACIÓN 39 WILLOW GARAGE. 56
ILUSTRACIÓN 40 COMPROBACIÓN DE LA INSTALACIÓN DE RVIZ. 58

Ilustración 41 roscore. 63
Ilustración 42 NAOqi preparado. 63
Ilustración 43 nao_bringup. 64
Ilustración 44 nao_description. 64
Ilustración 45 Nao en rviz. 65
Ilustración 46 rosservice. 65
Ilustración 47 Breve ejemplo de programa. 65
Ilustración 48 Uso del robot Pepper en Carrefour. 67
Ilustración 49 Creación de una cuenta en Aldebaran Robotics. 69
Ilustración 50 Diferentes posibilidades de descarga del instalador. 70
Ilustración 51 Comienzo del proceso de instalación. 70
Ilustración 52 Aceptación del contrato de licencia del software. 71
Ilustración 53 Inserción de la clave de licencia. 71
Ilustración 54 Elección del modo de instalación. 72
Ilustración 55 Instalación en curso. 72
Ilustración 56 Instalación finalizada. 73
Ilustración 57 Primera ejecución del programa. 73
Ilustración 58 Diferentes instaladores de Python NAOqi SDK. 74
Ilustración 59 Instalación de Python. 75
Ilustración 60 Selección de la ruta para la creación del directorio del programa. 75
Ilustración 61 Comienza el proceso de instalación. 76
Ilustración 62 Instalación completada. 76
Ilustración 63 Python en Windows. 77
Ilustración 64 Comienzo de la instalación de Python NAOqi SDK. 77
Ilustración 65 Selección del directorio de Python 2.7. 78
Ilustración 66 Instalación. 78
Ilustración 67 Instalación finalizada. 79
Ilustración 68 Comprobación de la instalación. 79
Ilustración 69 Diferentes instaladores de C++ NAOqi SDK. 81
Ilustración 70 Servidor operativo. 90
Ilustración 71 HttpRequester. 90
Ilustración 72 Esquema de comunicación. 92

1. INTRODUCCIÓN Y OBJETIVOS

En los últimos años, el modo de pensar de la sociedad ha evolucionado, tanto es así, que cuando vamos a comprar queremos ser alguien, es decir, no ser tratados por iguales y definidos como clientes. Queremos que se nos trate de un modo diferente a los demás, que las promociones que se nos muestren estén acordes a nuestras preferencias. En resumen queremos un servicio personalizado.

Como ejemplos claros de este trato hacia al cliente tenemos páginas web como la de Amazon o PcComponentes, las cuales te avisan de promociones y ofertas de productos que has comprado o similares.

Este tipo de trato nos encanta, pero aun así no tenemos suficiente, queremos que este modo de mejorar la experiencia a la hora de comprar siga evolucionando. En la búsqueda de esta mejora se ha ido incluyendo diversas ramas de conocimientos, siendo una de ellas la robótica, ampliando así los límites de este desarrollo.

Un aspecto importante a la hora de realizar todo este desarrollo es saber hasta dónde se puede llegar y que herramientas son las idóneas para alcanzar los objetivos iniciales propuestos. Todos conocemos lo atractivo que resulta un robot para cualquier persona, si además te ayuda y personaliza tu compra, esto hace que los posibles proyectos en los que se incluyan un robot resulten muy interesantes.

Ilustración 1 Robot NAO.

Por ello se ha introducido el uso del robot Nao en este proyecto. Este autómata desarrollado por la empresa francesa Aldebaran Robotics ha sido empleado en diferentes tareas desde su lanzamiento en 2004 y en 2007 sustituyó al robot Aibo como plataforma estándar para la Robocup ("Robot Soccer Wold Cup"). Aunque su versión destinada a la investigación y educación se retrasó hasta 2008, incluyéndose en universidades como la IIT de Kanpur de la India, la Universidad del Rey Fahd de Petróleo y Minerales de Arabia Saudita y la Universidad de Tokio. Finalmente en 2011 se lanzó su venta al público con una mejora de software y hardware, tales como cámaras de alta definición, mayor robustez o sistema anticolisión.

El robot está dotado de 14, 21 o 25 grados de libertad (GDL) dependiendo de la versión y viene con un paquete de software que incluye una herramienta gráfica de programación, Chroegraphe, un software de simulación y un kit de desarrollo de software.

Retomando una idea de un párrafo anterior, acerca de los límites de la mejora de la experiencia de compra, está la iniciación con ROS. Una vez que la empresa comercializó el autómata al público, empezó la colaboración entre diferentes miembros de la comunidad de ROS, creándose así una documentación con la que poder iniciarse a este framework. Aunque desde mi punto de vista, el precio del robot, actualmente unos 7.000 €, no es accesible para la mayoría de la sociedad, por ello la documentación puede resultar escasa y desactualizada. Este es uno de los motivos que no recomiendo comenzar directamente con ROS y Nao, ya que la herramienta es bastante compleja de manejar y si

sumados esta laguna de información acerca del robot tenemos como resultado un aprendizaje demasiado lento y con fisuras.

Resumiendo, las razones que explican la existencia de este TFG son las siguientes:

- Investigar que se puede hacer con el robot dentro del sector RETAIL.
- Aprendizaje de la herramienta ROS.
- Profundizar y desarrollar los conocimientos adquiridos durante mi formación como ingeniero.

Así, el propósito de este proyecto es realizar todo el software necesario para que el robot pueda comunicarse con una persona y ayudarle a realizar la compra. Por ello, los objetivos que tendremos que se establecer son:

- El robot reconocerá a las personas mediante un código QR.
- Envío de información entre el autómata y el servidor.
- Habilitar la posibilidad para que el robot pueda hablar y gesticular, siendo una comunicación más atractiva a las personas.
- Investigación del entorno del NAO en ROS.

Todos los objetivos anteriores fueron los requisitos propuestos por la empresa, solo el último fue un objetivo propio aconsejado por los profesores, ya que entienden que ROS será una herramienta muy poderosa en futuros proyectos.

Como es lógico todo el proyecto estará contenido dentro del ámbito del RETAIL. Este es un término anglosajón, el cual hace referencia a la venta minorista. Por ello cuando estamos hablando de retail nos referimos a la comercialización de productos al por menor.

Con ello se entiende que el objetivo del proyecto sea hacer que el robot sea atractivo e interesante para los diferentes clientes que visitaran el supermercado. Con esto se entiende toda la interacción gestual, y si a esto le sumamos que el robot hablará, la atención de los clientes quedara atraída hacia nuestro autómata.

Por otro lado, todo este proyecto sería inútil si no pudiésemos intercambiar información. Esta función la realizarán los servidores que se crearán, en el lado del robot y en el lado de la empresa, aunque este segundo no es nuestra obligación. Aunque se realizó un servidor que simulaba la tarea que tendría que realizar el servidor real.

En conjunto, el proyecto se compone de:

- El software del robot, desarrollado en Python y usando el sistema operativo Linux.
- El código del servidor del Nao, que nos ofrece la posibilidad de enviarle la información pertinente.
- El código del servidor de la empresa para poder simular todo el proyecto en su conjunto.

Puede parecer que el proyecto no tiene dificultad alguna, pero el código de los servidores fue realizado en Java debido a una imposición de la empresa. Esto significó un cierto retraso debido a mi desconocimiento acerca de este lenguaje de programación.

Además hay que tener en cuenta que el código del autómata está en Python con lo cual la forma de que "hablasen" el servidor del robot en Java con el software del robot en Python fue mediante la creación de sockets con su configuración posterior para su correcto funcionamiento.

Todos estos pequeños problemas se han solucionado creando un sistema robusto, cumpliendo todos los objetivos, haciendo que la ejecución de dicho programa sea un gran comienzo para el desarrollo de autómatas con programas de cara al público.

2. DESCRIPCIÓN DEL ROBOT NAO

A lo largo de todo este punto vamos a conocer las especificaciones técnicas del Nao, así como todos los actuadores y sensores que tienen el robot, los tipos de cuerpo que podemos encontrar y como diferenciarlos, una breve guía acerca de los primeros pasos con nuestro Nao y una guía detallada acerca de la conexión del robot a internet.

2.1 Lista de sensores y actuadores

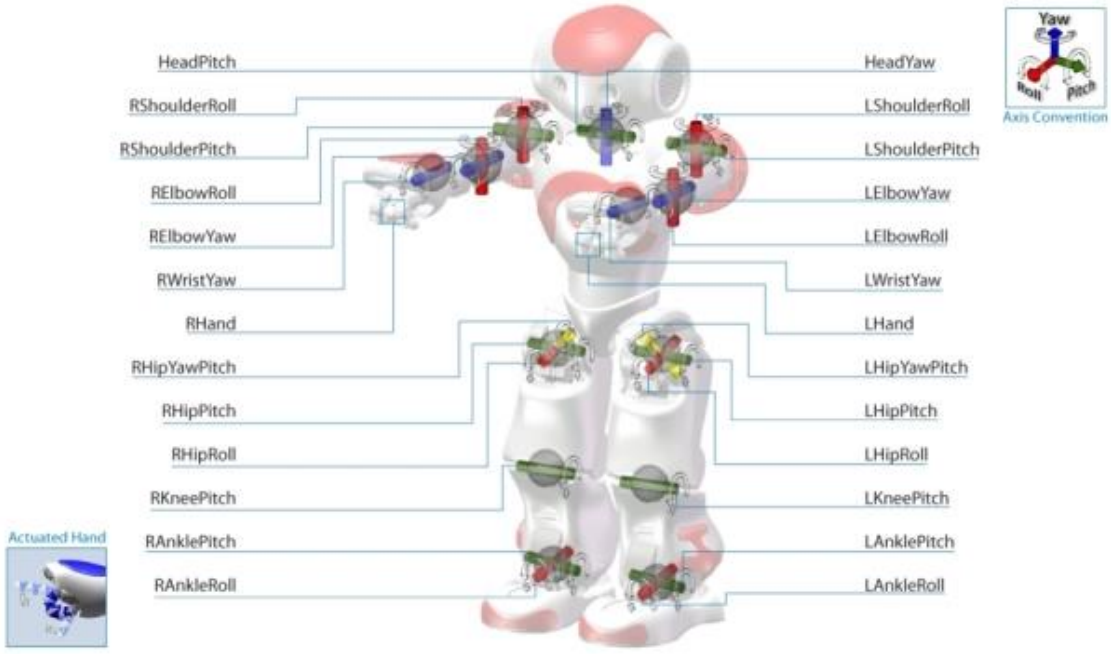


Ilustración 2 Lista de articulaciones y actuadores.

Cada actuador tiene 6 sensores:

- **Posición (radianes):** ángulo de la articulación.

- **Sensor de posición (%):** grado de apertura de las manos, 0 es cerrado, 1 abierto.
- **Intensidad del motor (A):** si esta intensidad es mayor que la máxima un PWM se encargará de disminuir dicha corriente y luego la aumentará. De esta forma, protege el motor, la electrónica interna y la parte mecánica de la articulación.
- **Temperatura (ºC):** esta es simulado utilizando el dato de la intensidad del motor. Existe un límite de temperatura para proteger al motor, este está impuesto por la versión del robot.
- **Estado de la temperatura con respecto al límite:** este dato puedo variar entre:
    - 0: temperatura normal.
    - 1: la temperatura ha llegado al máximo, comienza a reducirse la rigidez.
    - 2: calor excesivo en la articulación, rigidez reducida un 30%.
    - 3: calor crítico en la articulación, el valor de rigidez es 0.
- **3 más para notificarnos algún error:** estos sensores dependen del módulo "ALDiagnosis" y "Temperature Diagnosis". Existen tres niveles posibles, "**NEGLIGIBLE**", "**SERIOUS**" o "**CRITICAL**".
    - **NEGLIGIBLE o INSIGNIFICANTE:** Sin efecto para el robot.
    - **SERIOUS o SERIO:** El control de rigidez de la cadena que incluye la articulación o el actuador que falla es deshabilitado y no será posible su reactivación.
    - **CRITICAL o CRITICO:** El robot toma la posición de descanso y se niega a reactivarse. El robot queda inútil.

2.2 Versión y tipo de cuerpo

Existen formas de conocer qué tipo de cuerpo y versión tiene nuestro robot. Para conocer exactamente que versión tiene nuestro robot nos podemos fijar en la parte posterior de su cabeza:

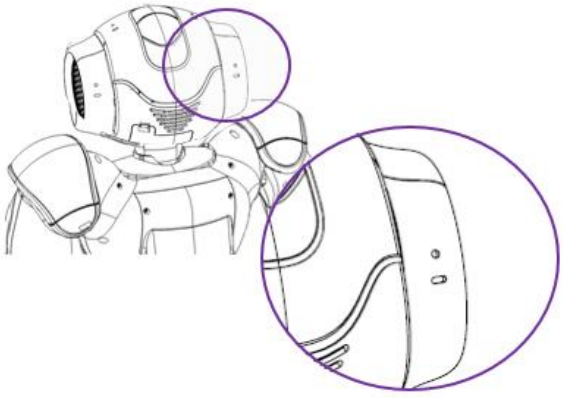


Ilustración 3 Versión V5.

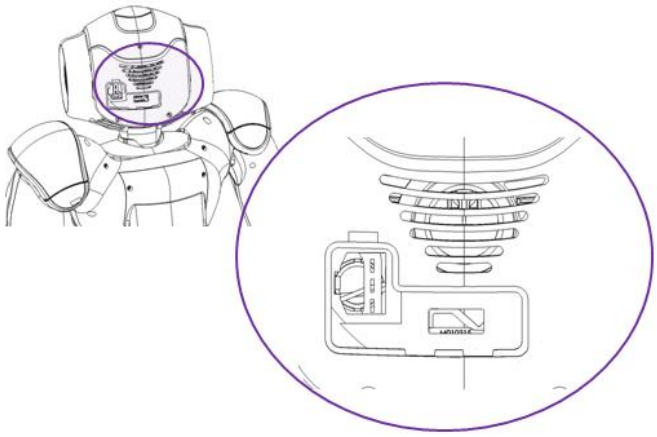


Ilustración 4 Versión V4.

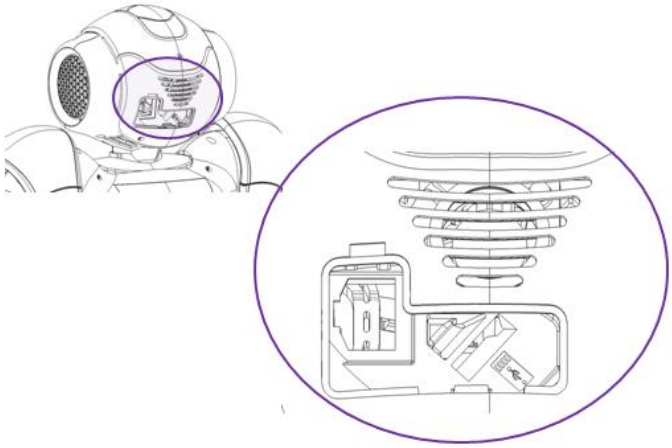


Ilustración 5 Versión V3.3.

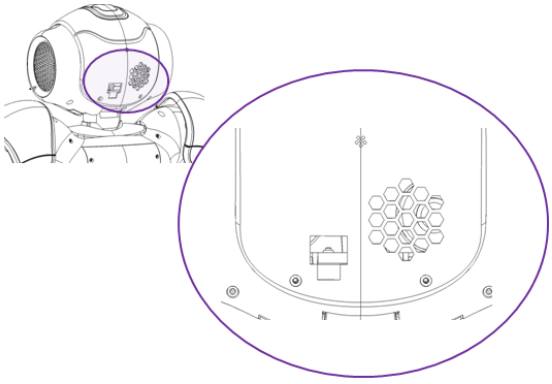


Ilustración 6 Versión V3+, V3.2.

Otra característica es el tipo de cuerpo de nuestro robot:

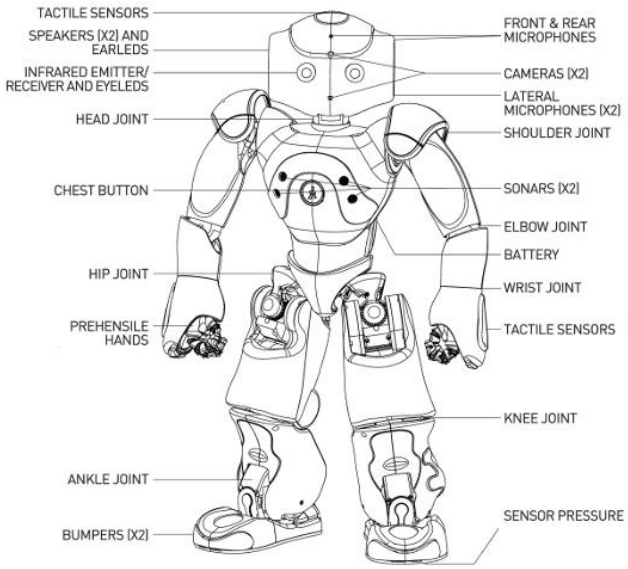


Ilustración 7 Tipo de cuerpo H25.

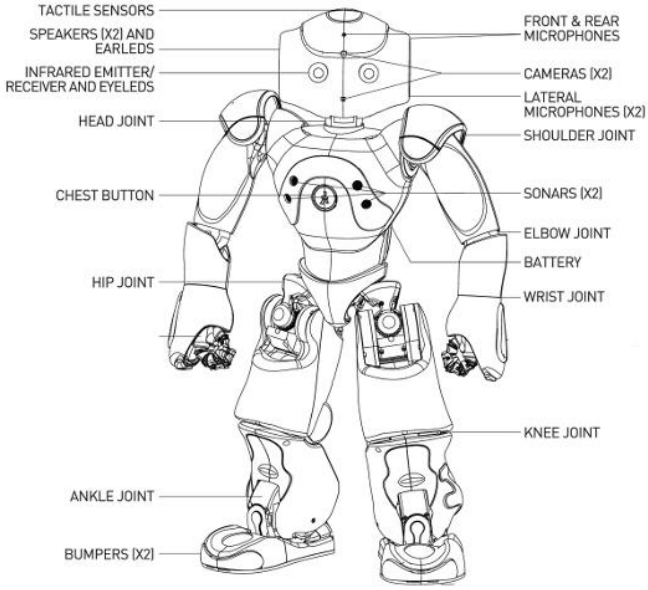


Ilustración 8 Tipo de cuerpo H21.

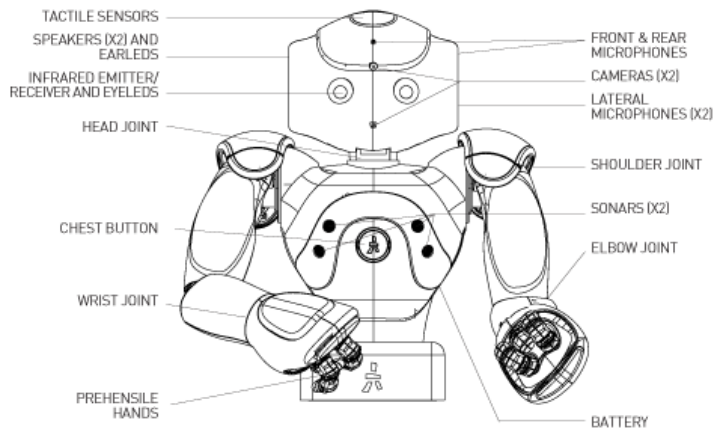


Ilustración 9 Tipo de cuerpo T14.

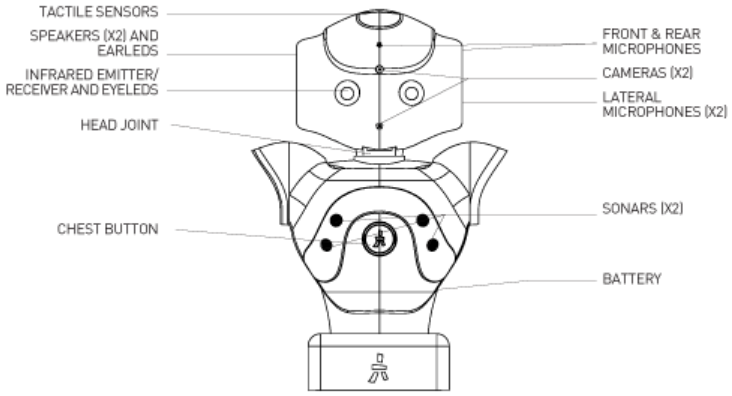


Ilustración 10 Tipo de cuerpo T2.

Realmente conocer el tipo de cuerpo nos aporta la información de conocer cuántos grados de libertad posee nuestro autómata, los tipos T2 y T14 tienen 14 GDL, el H21 tiene 21 GDL y H25 tiene 25. Aun así, si queremos saber exactamente qué tipo de cuerpo y versión tiene nuestro robot también podemos usar un programa en Python que dejare en el apartado "Anexos".

2.3 Fuera de la caja

Lo primero que se debe hacer, si no se ha usado nunca el NAO, es leerse la guía de seguridad que viene dentro de la caja.

En esta guía de seguridad podemos encontrar detalles como la garantía del robot, la manipulación y alimentación del mismo y los aspectos que tendremos que tener en cuenta para que el robot no sufra ningún daño.

El siguiente paso es sacar el robot de la caja, con sumo cuidado procedemos a la operación, teniendo especial cuidado en las partes frágiles del robot y en no sufrir ningún accidente ya que es probable que al desembalar el robot nos pellizquemos las manos con las articulaciones.

Una vez sacado de la caja es conveniente colocar el NAO en una posición de seguridad como la que se muestra en la imagen.

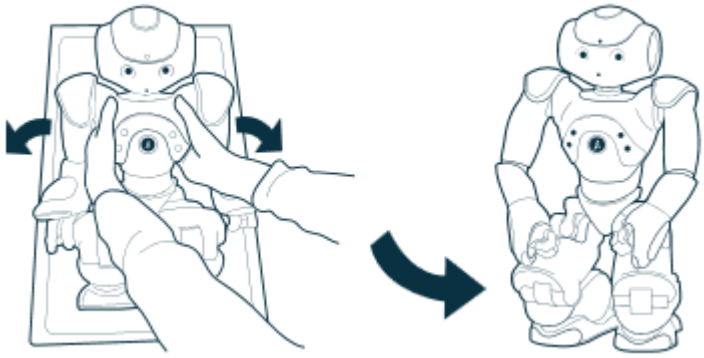


Ilustración 11 Posición de seguridad del NAO.

A continuación, deberemos alimentar la batería del NAO, para ello tendremos que utilizar que viene en la caja. Si echamos un vistazo a la espalda del robot en la parte inferior localizaremos la entrada del conector para cargar la batería, conectando el otro extremo directamente al enchufe (no desde un alargador como podemos leer en la guía de seguridad).

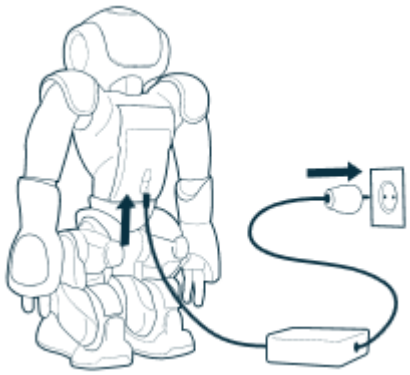


Ilustración 12 Conexión de la batería.

El siguiente paso será proveer de Internet al robot, para ello en la parte posterior de la cabeza encontramos una tapa, removiendo esta, encontraremos el puerto donde tendremos que conectar el cable Ethernet.

Ilustración 13 Conexión del Ethernet.

El otro extremo del cable Ethernet podremos conectarlo a nuestro ordenador, por ejemplo. El montaje tendría que quedar tal que así.

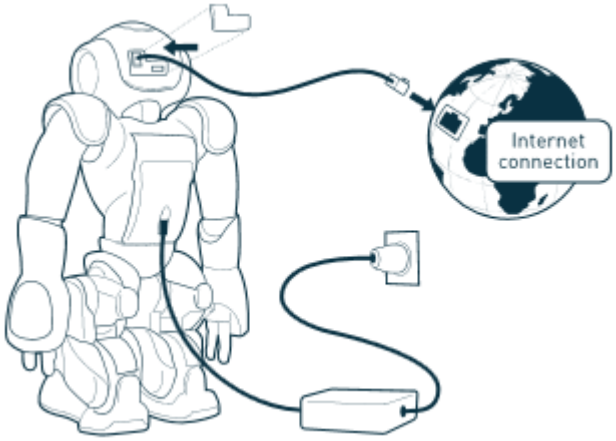


Ilustración 14 Finalización de montaje.

Llega el momento de encender por primera vez el robot. Para ello presionaremos el botón, durante 1 segundo, que encontramos en el pecho del mismo con el logo de Aldebaran Robotics. Luego tendremos que esperar entre 3-5 minutos, cuando el robot termine el proceso de encendido dirá "Ognak gnouk".

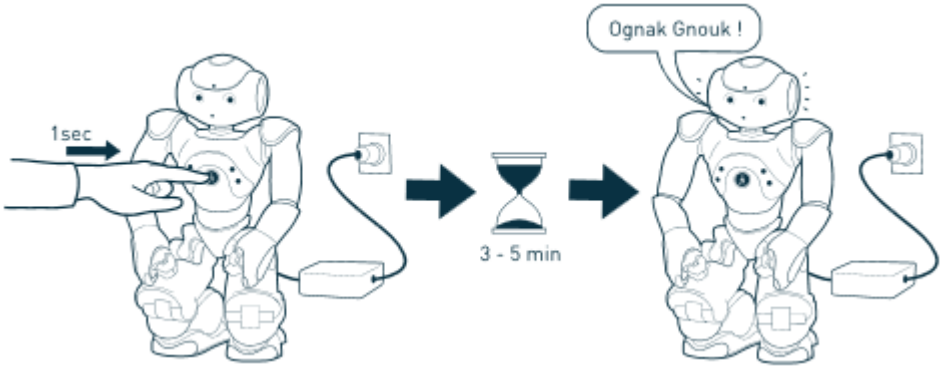


Ilustración 15 Proceso de encendido.

2.4 Primera configuración

2.4.1 Conectividad

Con el robot encendido y con el montaje con el cable Ethernet hecho, presionamos el botón que tiene el robot en el pecho, acto seguido el NAO nos dirá su dirección IP, es muy útil tener lápiz y papel preparado para anotar este número. El siguiente paso sería abrir en nuestro ordenador el navegador de nuestra elección y en la barra de direcciones escribir el número que nos ha proporcionado el robot.

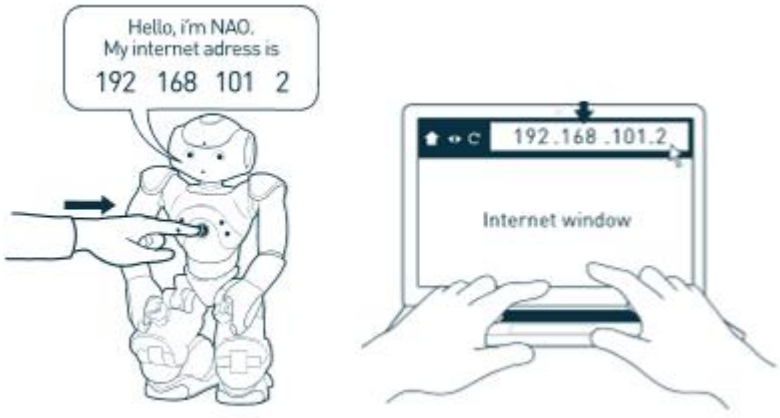


Ilustración 16 IP del robot.

Llegaremos a la página de personalización y configuración del NAO, pero antes de ser capaces de realizar cualquier modificación tendremos que logearnos dentro de esta interfaz, para ello tanto nombre como contraseña serán “nao”.

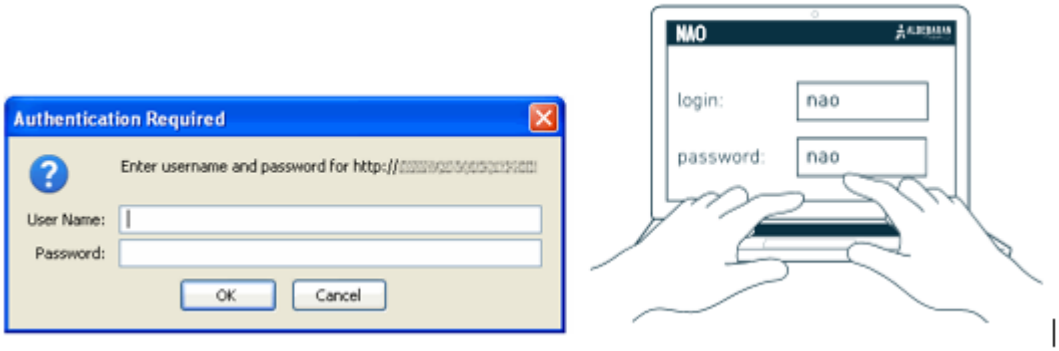


Ilustración 17 Proceso de acceso.

2.4.2 Selección del idioma

Una vez realizada la conectividad, la primera vez que lo hagamos, nos aparecerá esta ventana.

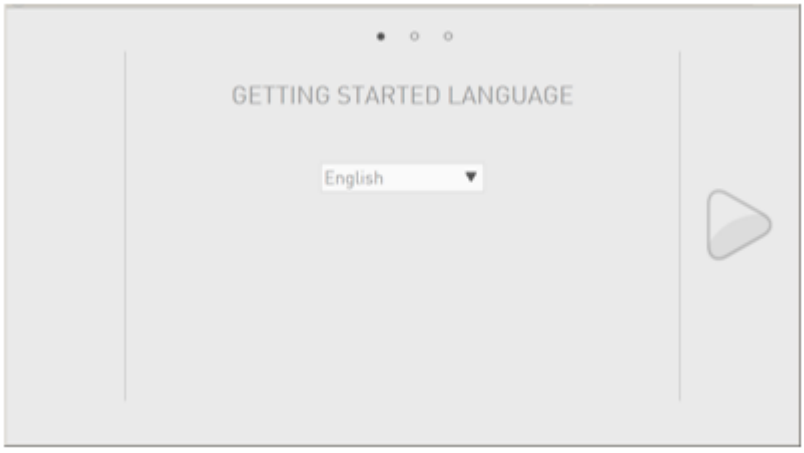


Ilustración 18 Asistente de configuración.

Este es el asistente de configuración de todas las opciones del NAO que podemos modificar. Lo primero que tendremos que seleccionar es el idioma que usará nuestro robot. A continuación, hacemos clic en la flecha, lo siguiente que nos aparecerá en pantalla son las condiciones que tenemos que validar en el contrato de licencia del software.

Ilustración 19 Condiciones del contrato de licencia

2.4.3 Configuración de la conexión WiFi

Antes hemos realizado la conexión del robot mediante Ethernet, pero esta práctica no es muy cómoda a la hora de trabajar con el robot ya que impide la libertad de movimientos del robot, esta es la razón por la cual vamos a pasar a la conexión WiFi.

Lo primero que tendremos que hacer es buscar la opción para ver todas las redes a las que el robot podría acceder. Una vez en esta pantalla, deberemos seleccionar la red a la que queramos conectarnos.

Ilustración 20 Ejemplo de redes

Hacemos clic en la red deseada, automáticamente nos saltará un menú que nos pedirá información acerca de la red, aunque este menú nos presenta la posibilidad de acceder a opciones avanzadas nosotros nos centraremos en la contraseña. Una vez rellenado este campo presionamos "connect".

Ilustración 21 Configuración de la red deseada.

2.4.4 Personalización del robot

Si navegamos un poco por la interfaz que se nos muestra en nuestro navegador, encontraremos una opción para cambiar el nombre al robot. Esto es útil si estamos trabajando con más de un robot.

Una vez que llegamos a la opción de cambiar el nombre, los pasos a seguir son bastante sencillos. Lo primero es escribir el nuevo nombre que deseamos para nuestro robot.

Ilustración 22 Cambio de nombre

También podemos modificar la contraseña, aunque esta opción no la recomiendo ya que la contraseña será necesaria para conectarnos al robot y a su página web. El proceso para hacerlo es muy sencillo, solo hay que indicar cuál es la nueva contraseña y validarla luego.

Ilustración 23 Cambio de contraseña

Nota: Es de vital importancia guardar la contraseña ya que esta será requerida para la conexión a la página web del robot.

A continuación, podemos realizar el acceso a Aldebaran Cloud, solo tenemos que rellenar los campos de "*Login*" y "*Password*" o "*Nombre de usuario*" y "*Contraseña*".

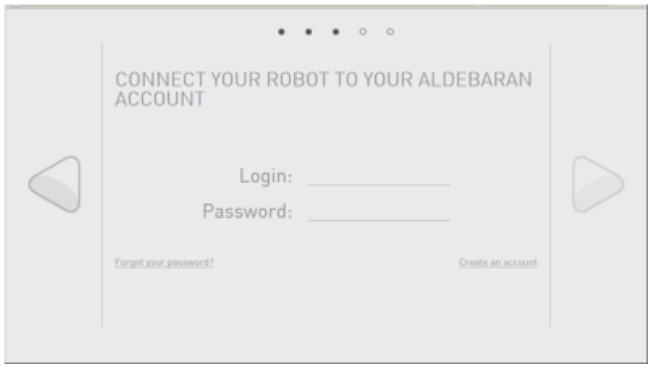


Ilustración 24 Acceso a Aldebaran Cloud.

En este punto podemos encontrarnos en uno de estas dos opciones:

1- La personalización del nombre del robot se realizó en pasos anteriores:

Ilustración 25 Mensaje de aviso para validar la configuración.

2- No se personalizó el nombre del robot en pasos anteriores:

Ilustración 26 Proceso finalizado.

2.4.5 Mi conectividad

2.4.5.1 Windows

Debido a la particularidad de la red que usé, tuve que hacer una primera configuración para poder tener la opción de tener conexión inalámbrica con el robot.

La red empleada es la proporcionada por la universidad, eduroam, pero para acceder a esta red hacen falta unas credenciales, como usuario de uvus y contraseña. Necesitaba la conexión a internet en mi portátil y en el robot, ya que si quieres que el robot realice algún movimiento es más seguro si no está limitado por la longitud del cable, todo este problema me surgió al principio del proyecto cuando estaba usando Windows. La idea para resolver este problema fue bastante sencilla, crear un punto de acceso con mi ordenador y compartir la conexión a internet.

Para ello usaremos el terminal en Windows, para abrir el símbolo del sistema podemos escribir cmd en el inicio y pulsar enter, otra forma es la tecla Windows + R, escribir cmd y pulsar enter.

Una vez que no encontremos ante el símbolo del sistema de Windows solo tendremos que teclear 2 comandos.

```
Netsh wlan set hostednetwork mode=allow ssid=NombreRed key=contraseña

Netsh wlan start hostednetwork
```

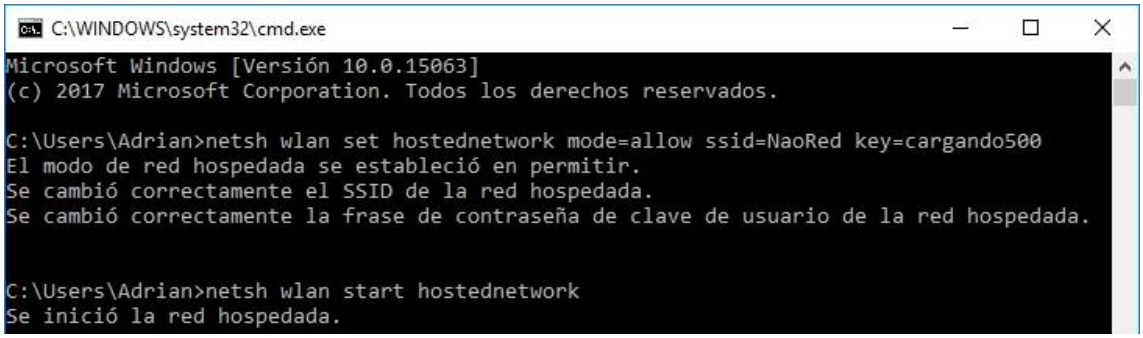


Ilustración 27 Creación de la red hospedada en Windows.

A continuación, tenemos que dirigirnos a “Conexiones de red”, si tenemos Windows 10 podemos usar a Cortana para encontrarlo más fácil, otra forma es tecla de Windows + X y hacer clic en “Conexiones de red”.

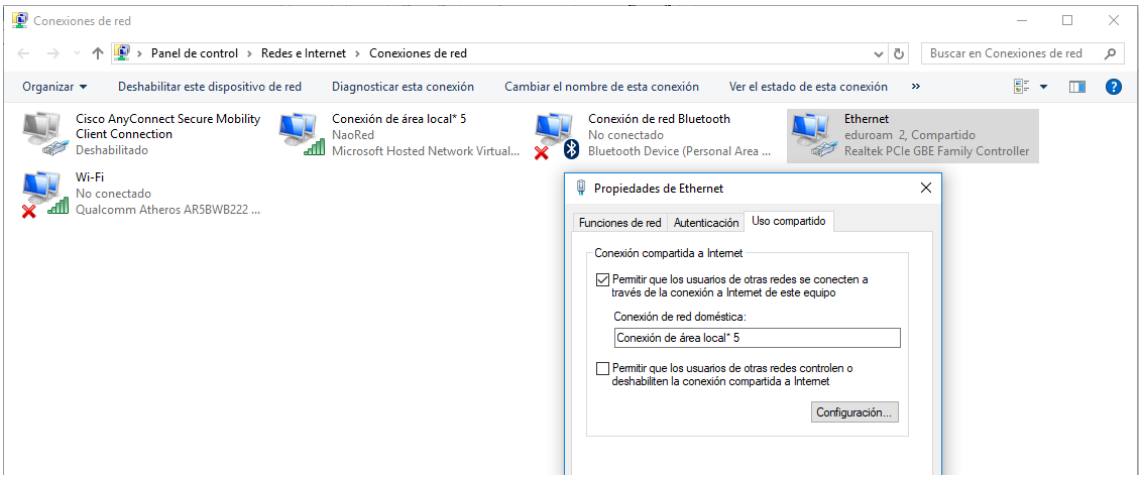


Ilustración 28 Proceso para compartir internet a la red hospedada.

Y en la red que estamos usando (no la que hemos creado en los pasos anteriores) hacemos clic derecho, seleccionamos en "Propiedades", hacemos clic en la pestaña "Uso compartido", elegimos la primera opción. Por último, tenemos que especificar la red con la que vamos a compartir la conexión a internet. En el ejemplo es "Conexión de área local* 5".

Una vez realizada toda esta configuración, el robot podrá ver esta red, para que el autómata tenga internet solo tenemos que irnos a la página web del robot y entre las redes visibles estará la nuestra, haciendo clic en ella y rellenando el campo de la contraseña tendríamos conexión WiFi en nuestro robot.

Para acabar cuando queramos cerrar esta red solo tendremos que introducir el siguiente comando:

```
Netsh wlan stop hostednetwork
```

Otra forma bastante más fácil para realizar todo este proceso es encontrar el icono de red en la barra de herramientas, en la esquina inferior derecha  . A continuación, doble clic en la red a la que estemos conectados y nos aparece la siguiente ventana.

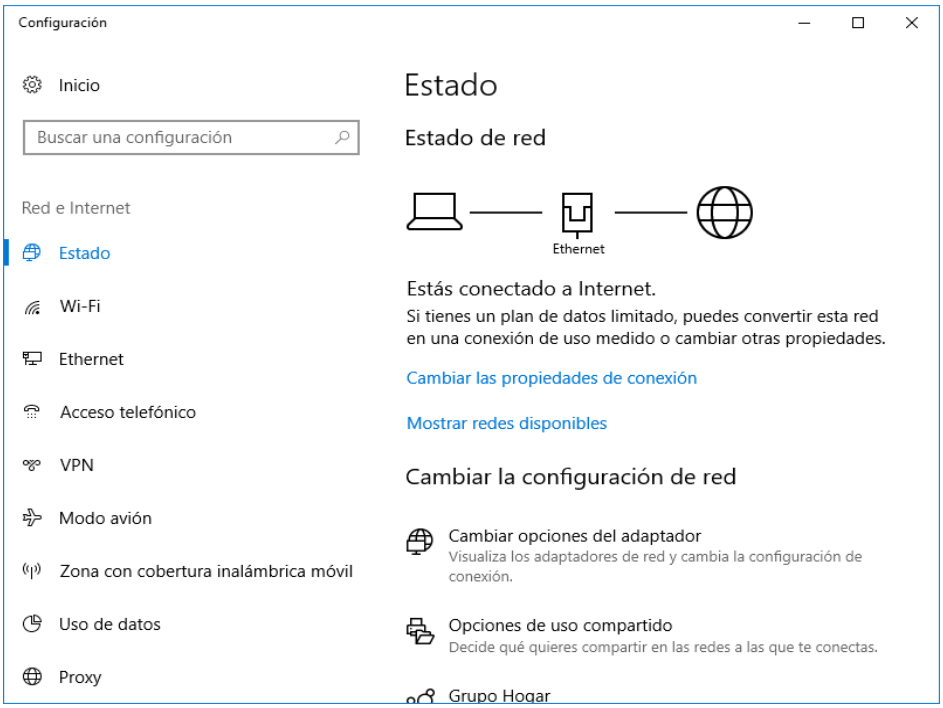


Ilustración 29 Menú de configuración de Windows 10.

Seleccionamos "Zona con cobertura inalámbrica móvil", haciendo clic en el botón "Editar", podemos modificar el nombre y la contraseña de la red, debemos seleccionar la red a la que estemos conectados para compartir la conexión.

Por último, para habilitar la nueva red solo debemos hacer clic en el botón deslizable que encontramos en la parte superior. Además, en futuras conexiones no tendremos que hacer ninguna modificación adicional, solo activar la red cuando la vayamos a usar.

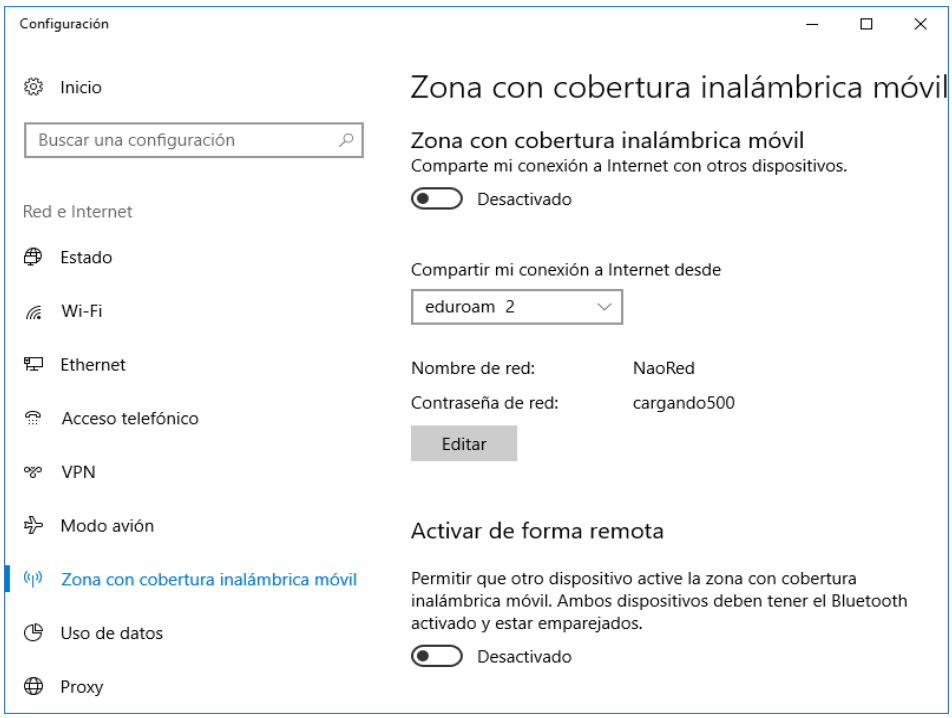


Ilustración 30 Creación de la red hospedada de forma sencilla.

2.4.5.2 Ubuntu

Para hacer un punto de acceso en Ubuntu solo hay que seguir los siguientes pasos:

Una vez que estemos en el escritorio, arriba a la derecha nos encontramos el icono de la conexión de red a la izquierda del icono Es, el icono que es un radar. Hacemos clic en ese icono y buscamos la opción "Crear una red inalámbrica nueva…"

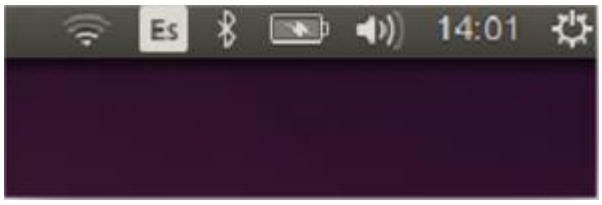


Ilustración 31 Panel superior de Ubuntu 14.04.

Nos aparecerá la siguiente ventana:

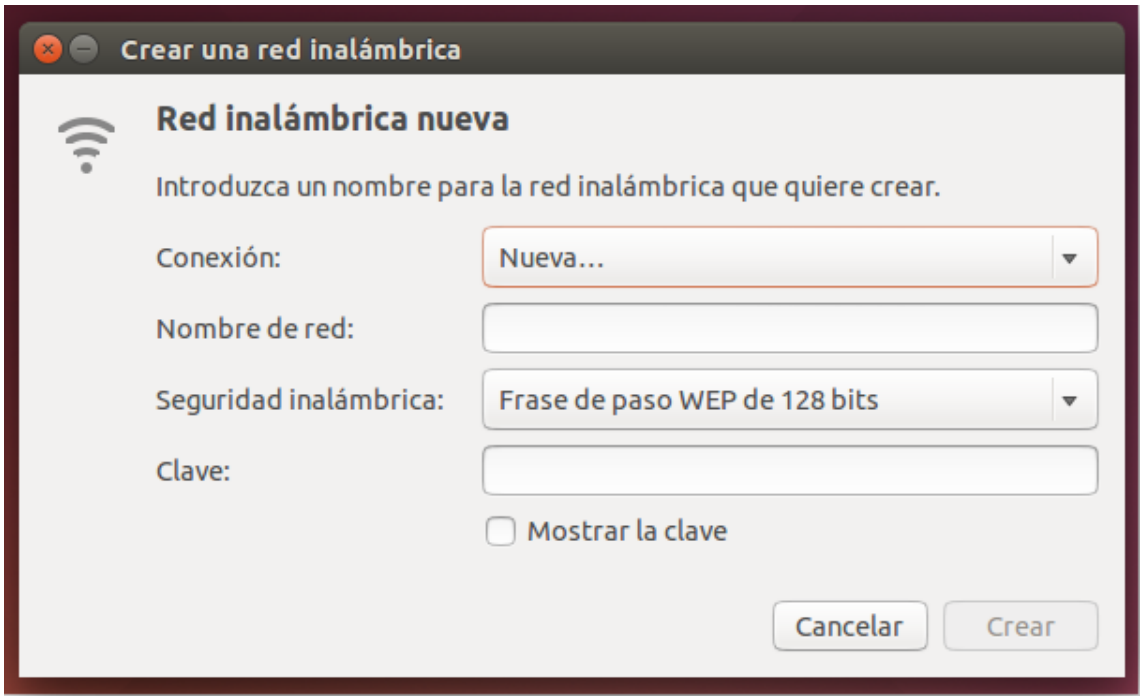


Ilustración 32 Menú de configuración de la red.

Para crear la red deberemos completar los siguientes campos:

- Nombre de red: el nombre con el que identificaremos nuestra red.
- Seguridad inalámbrica: el tipo de seguridad que queremos aplicar. Recomiendo WPA y WPA2.
- Contraseña: la clave de acceso para evitar que cualquier persona se conecte a nuestra red.

Una vez completados estos campos, hacemos clic sobre el botón "Crear" y automáticamente Ubuntu nos creará nuestro punto de acceso inalámbrico. Si nos fijamos en la parte superior derecha (misma zona que la primera imagen) el icono habrá cambiado al siguiente.

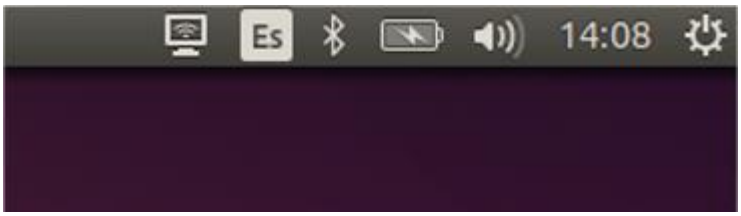


Ilustración 33 Comprobación de la creación de la red.

Lo único que nos quedará por hacer será entrar en la página web del robot y acceder a esta red.

Nota: Aunque describo como hacer el punto de acceso, en los laboratorios la conexión inalámbrica no era estable con lo cual cuando usaba Ubuntu prefería tener al robot conectado mediante una conexión cableada a mi portátil (cable Ethernet) para eliminar la duda del comportamiento del robot al perder la conexión.

3. Entornos de programación NAO

El robot Nao admite varias plataformas de programación que nos proporcionan diferentes formas de enfocar los problemas y diversos caminos por los que optar según los requisitos que nos hayan impuesto. Explicare parte de la evolución del lenguaje si procede o si puedo encontrar información acerca de él y realizaré una explicación breve acerca de los que nos puede aportar.

3.1 Choregraphe

Choregraphe es una aplicación de escritorio multiplataforma, que nos permite:

- Crear animaciones, comportamiento y diálogos,
- Probarlos en un robot simulado, o directamente en uno real
- Monitorizar y controlar tu robot
- Enriquecer comportamientos en Choregraphe con tu propio código en Python

Choregraphe nos permite crear aplicaciones que contengan diálogos, servicios y comportamientos interesantes, como interacción con personas, bailar, enviar e-mails, sin escribir ni una línea de código.

De todos los entornos de programación que podemos emplear para comunicarnos con el robot, Choregraphe es el más sencillo.

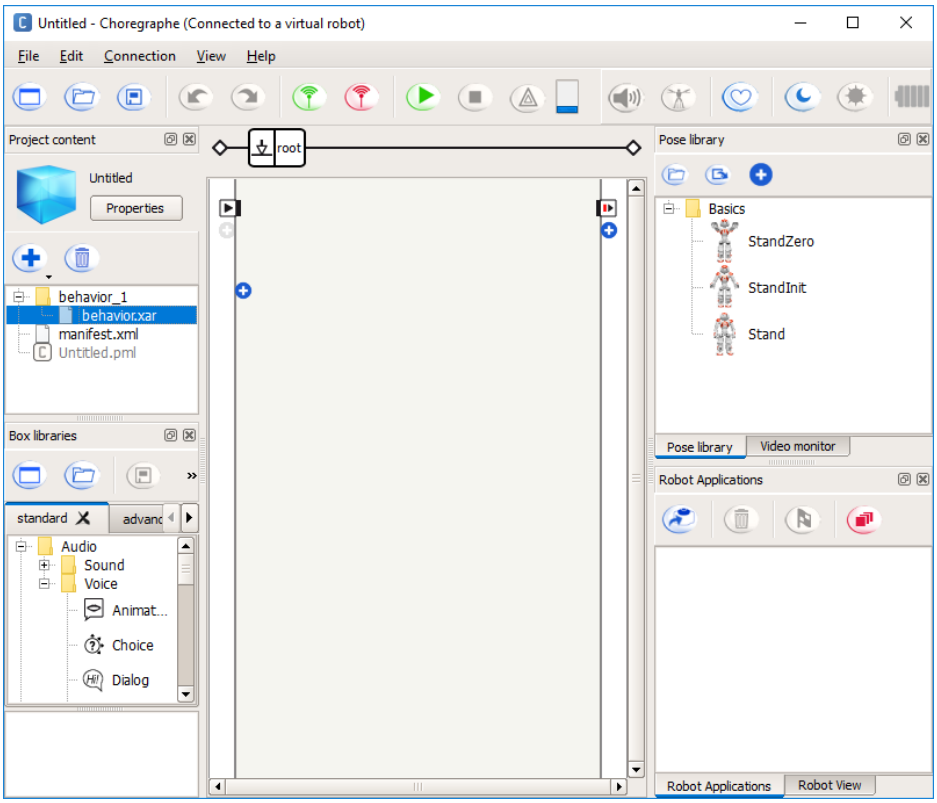


Ilustración 34 Choregraphe.

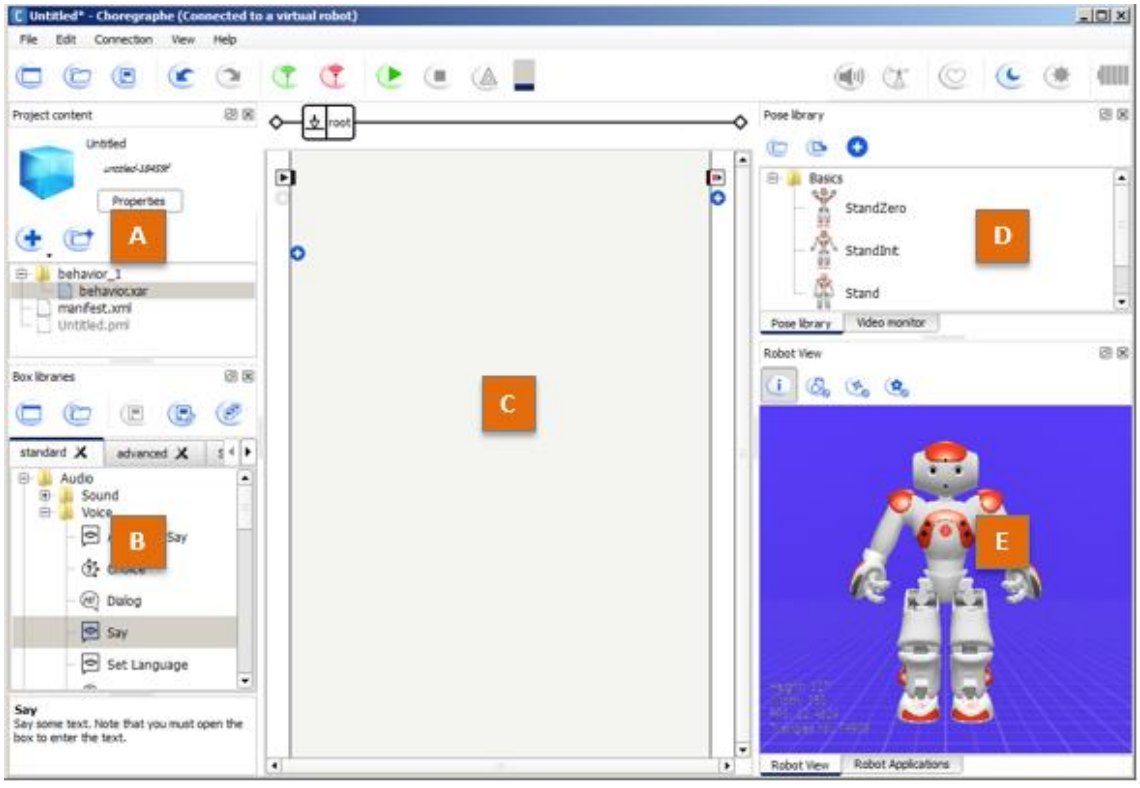


Ilustración 35 Distribución de los paneles de Choregraphe.

La organización de paneles del programa es la siguiente:

- A: Panel de contenido del proyecto.
- B: Panel de librerías de cajas.
- C: Panel de diagrama de flujo.
- D: Panel de librería de posición y monitor de video.
- E: Vista del robot y panel de aplicaciones del robot.

Como podemos ver en el panel marcado con la "*B*" encontramos un conjunto de cajas predefinidas con las cuales podemos hacer que el robot describa algún movimiento hasta establecer una conversación. Para ello o único que tendríamos que hacer es seleccionar una de las cajas que encontramos en "B" y arrastrarlos hasta "C", conectar las entradas y las salidas de la caja con los símbolos de la derecha y la izquierda del panel. Por último, solo tendríamos que pulsar el botón verde de "Reproducir" o "Play".

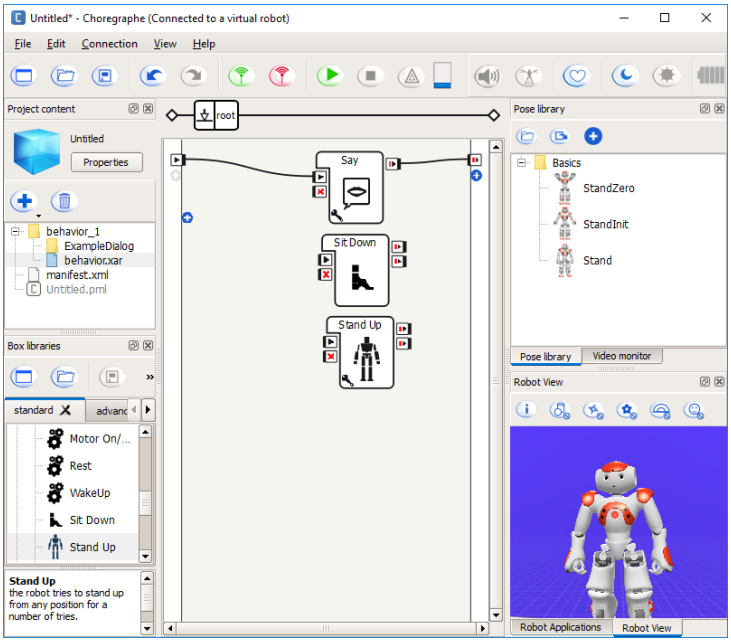


Ilustración 36 Breve ejemplo de creación de un programa.

Como podemos comprobar este entorno de programación es el más amistoso para el usuario, aunque este no domine los conceptos básicos de programación. Choregraphe es el software que nos proporciona Aldebaran Robotics para comenzar a desarrollar proyectos que involucren al NAO. Aunque he dicho antes que esta interfaz de programación es bastante básica se puede ir un poco más allá, si hacemos doble clic en las cajas podemos ver que, en el fondo, están creadas con Python.

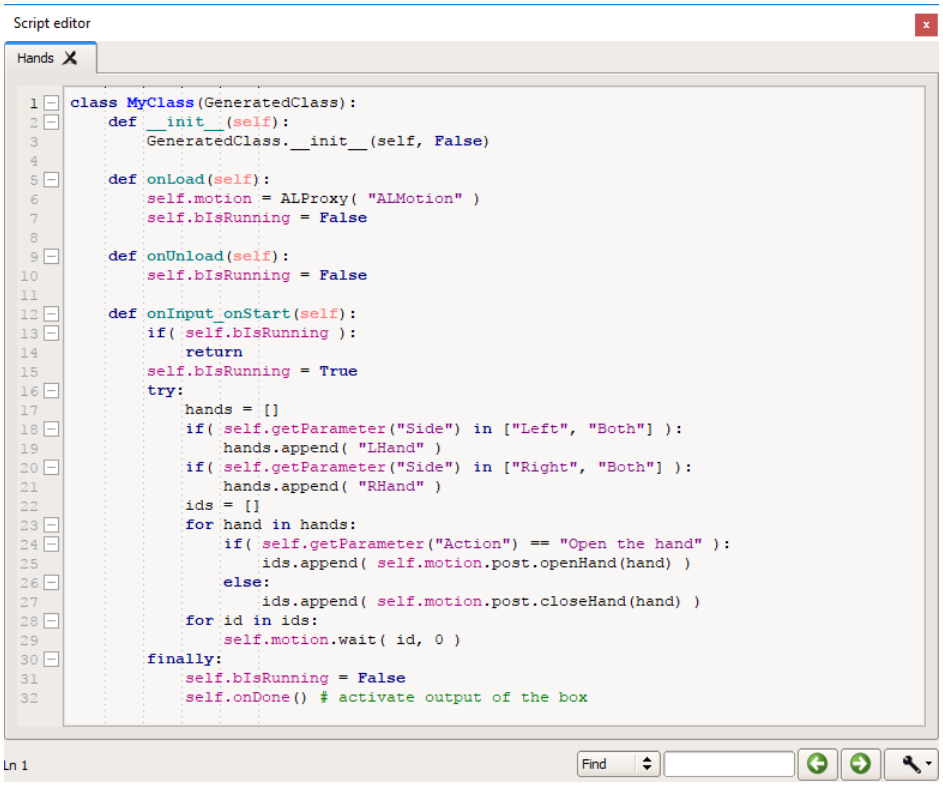


Ilustración 37 Contenido del interior de la cajas.

Siguiendo esta estructura podemos crear cajas que usen métodos ya definidos anteriormente en los cuales podemos realizar las acciones que queramos. Existe documentación online que explica cómo usar este tipo de programación, aun así al combinar varias formas de programación se deben cumplir todos los requisitos para que el robot describa la actividad en cuestión.

Tanto es así que para que la caja que hemos creado funcionase debemos cumplir varias reglas:

- Cumplir todas las convenciones respecto al lenguaje de programación Python.
- Ceñirnos a las estructuras que han sido creadas previamente, es decir, tenemos que usar los métodos que podemos encontrar en la documentación.
- Respectar el modo de conexionado de las cajas dentro de la interfaz.

Personalmente veo esta forma de crear cajas, un poco absurda, ya que, si quiero crear mis propias cajas porque no usar directamente el framework PythonSDK y crear todo desde cero, creando mis propios métodos con mis propias reglas. Aun así entiendo que este framework es el mejor en cuanto relación programación/actividad, quiero decir, no es esencial que sepas algo sobre programación para hacer unas animaciones o unos diálogos elaborados.

3.2 Python y Python SDK

Python fue creado a finales de los ochenta por Guido Van Rossum en el Centro para las Matemáticas y la Informática en los Países Bajos, como un sucesor del lenguaje de programación ABC, capaz de manejar excepciones e interactuar con el sistema operativo Amoeba.

Van Rossum es el principal autor de Python, y su continuo rol central en decidir la dirección de Python es reconocido, refiriéndose a él como Benevolente Dictador Vitalicio.

En el año 2000, el equipo principal de desarrolladores de Python se cambió a BeOpen.com para formar el equipo BeOpen PythonLabs. CNRI (Corporation for National Research Initiatives) pidió que la versión 1.6 fuera pública, continuando su desarrollo hasta que el equipo de desarrollo abandonó CNRI; su programa de lanzamiento y el de la versión 2.0 tenían una significativa cantidad de traslapo.9 Python 2.0 fue el primer y único lanzamiento de BeOpen.com. Después que Python 2.0 fuera publicado por BeOpen.com, Guido van Rossum y los otros desarrolladores de PythonLabs se unieron en Digital Creations.

Python 2.0 tomó una característica mayor del lenguaje de programación funcional Haskell: listas por comprensión. La sintaxis de Python para esta construcción es muy similar a la de Haskell, salvo por la preferencia de los caracteres de puntuación en Haskell, y la preferencia de Python por palabras claves alfabéticas. Python 2.0 introdujo además un sistema de recolección de basura capaz de recolectar referencias cíclicas.

Python 2.1 fue un trabajo derivado de Python 1.6.1, así como también de Python 2.0. Su licencia fue renombrada a: Python Software Foundation License. Todo el código, documentación y especificaciones añadidas, desde la fecha del lanzamiento de la versión alfa de Python 2.1, tiene como dueño a Python Software Foundation (PSF), una organización sin ánimo de lucro fundada en el año 2001, tomando como modelo la Apache Software Foundation.1 Incluido en este lanzamiento fue una implementación del scoping más parecida a las reglas de static scoping (del cual Scheme es el originador).

Una innovación mayor en Python 2.2 fue la unificación de los tipos en Python (tipos escritos en C), y clases (tipos escritos en Python) dentro de una jerarquía. Esa unificación logró un modelo de objetos de Python puro y consistente.11 También fueron agregados los generadores que fueron inspirados por el lenguaje Icon.

Las adiciones a la biblioteca estándar de Python y las decisiones sintácticas fueron influenciadas fuertemente por Java en algunos casos: el package logging introducido en la versión 2.3, está basado en log4j; el parser SAX, introducido en 2.0; el package threading cuya clase Thread expone un subconjunto de la interfaz de la clase homónima en Java.

Se mejoró el manejo numérico, tanto para los números de punto flotante como para la clase "Decimal". Existen algunas adiciones útiles a la biblioteca estándar, como un módulo de prueba de unidades muy mejorado, el módulo argparse para analizar opciones de línea de comandos, las clases OrderedDict y Counter convenientes en el módulo de colecciones y muchas otras mejoras.

Python 2.7 está previsto para ser el último de los lanzamientos 2.x. Para ayudar a cambiar a Python 3, se han incluido varias nuevas características de la serie Python 3.x en 2.7.

Con la idea comentada en el otro apartado y con el deseo de empezar en el sistema operativo Ubuntu, cambié a este framework. Considero que esta herramienta es una de las mejores, ya que, la programación dentro del entorno de Python es bastante sencilla, por otra parte, el hecho de tener que programar en otro sistema operativo nos proporciona una mayor anchura de miras.

La configuración usada con este framework era la siguiente:

- Uso de gedit para escribir los programas. Aconsejo usar algún otro IDE que puedas configurar para programar Python, como Eclipse.
- Uso del terminal de Ubuntu para ejecutar los programas.
- PythonSDK como traductor de los programas hacia el robot.

En la página de Aldebaran Robotics podemos encontrar una breve introducción a cómo usar Python para programar el robot, aunque no es muy extensa, es suficiente para entender cómo usar todas las estructuras por defecto y si a esto le sumamos que Python es la forma más fácil de programación, obtenemos una forma de programar rápida y sencilla.

3.3 C++ y C++ SDK

C++ es un lenguaje de programación diseñado a mediados de los años 1980 por Bjarne Stroustrup. La intención de su creación fue el extender al lenguaje de programación C mecanismos que permiten la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido.

Posteriormente se añadieron facilidades de programación genérica, que se sumaron a los paradigmas de programación estructurada y programación orientada a objetos. Por esto se suele decir que el C++ es un lenguaje de programación multiparadigma.

Este es el framework más difícil de usar ya que más posibilidades permiten a la hora que crear tareas. La API de C++ es casi igual que en Python. Personalmente no he tocado este SDK aunque leyendo sus especificaciones podemos contar desde el principio con un mayor despliegue de opciones aunque la dificultad incrementa en cuanto a la programación. Aunque en la parte de aprendizaje de ROS, sí que toque este lenguaje de programación.

4. Desarrollo del proyecto

A lo largo de la ejecución de este proyecto me he encontrado problemas que he tenido que resolver. Describir todos los problemas y soluciones uno a uno sería una tarea pesada e infructuosa, así que solo citaré aquellos que crea importante.

4.1 Evolución del proyecto

Mis conocimientos al comienzo del proyecto eran los siguientes:

- Conocimientos en C y programación en MATLAB.
- Conocimientos en robótica por las asignaturas de Fundamentos de Robótica y Control y Programación de Robots.
- Conocimientos de redes por la asignatura de Arquitectura de Redes.

El inicio del proyecto trabajando con Choregraphe. Aunque este software puede ser bastante útil para alguien que no tenga demasiada idea acercar de programar robots, personalmente, esta forma de programación es bastante básica con lo cual prefiero hacer mi propio programa con mis funciones, las cuales realizarán las acciones que buscamos sin necesidad de seguir una estructura rígida para crearlas.

Por ello, el siguiente entorno de programación usado fue Python. Este es un lenguaje de programación bastante sencillo si lo comparamos con C++, pero aun así, se puede conseguir unos proyectos interesantes sin necesidad de complicarse demasiado. Debido a la baja complejidad de Python, experimente varios módulos del robot (ALAutonomousLife, ALMemory, ALMotion, ALAnimatedSpeech…).

Una vez entendido como usar los diferentes módulos del robot, pasé al desarrollo del programa principal del proyecto. El desarrollo del mismo no fue excesivamente complejo, ya que dividí su elaboración en diferente fases en la cuales iba consiguiendo diferentes objetivos.

Con la consecución de este objetivo, pasé a iniciarme en el entorno de ROS. El primer paso fue trabajar en los tutoriales que podemos encontrar en la página web de ROS y mediante su ejecución fui conociendo todos los conceptos que conforman este entorno.

El siguiente paso fue la mejora del nodo de exploración del Turtlebot. Durante el desarrollo de esta mejora, aprendí mucho más que solo siguiendo los tutoriales, este aprendizaje fue impulsado por 3 compañeros que tenían experiencia con este software. Con todo esto y en unas semanas conseguí desarrollar la mejora de navegación del Turtlebot. En este punto el lenguaje de programación era C++ y usaba POO (Programación Orientada a Objetos). A partir de aquí comencé a usar ROS con el NAO,

utilizando los diferentes nodos que estaban habilitados en el modelo de simulación. Aun así, ya estaba al final del proyecto, con lo cual el desarrollo de esta última parte no fue tanto como me hubiese gustado.

4.2 Problemas

Doble lectura del código QR: Debido a que el NAO trabaja por eventos, es decir, el módulo de lectura de QR estará siempre activado, con lo cual el robot leerá el contenido del código que le mostremos de forma continua. Esto no es un problema en sí, pero si le programamos que realice una acción cuando lea el QR, al realizar lecturas sucesivas, almacenará en memoria la ejecución de las acciones programadas anteriormente aunque el QR no haya cambiado.

Para un mejor entendimiento pasaré a explicarlo de otra forma, una vez el NAO detecte que tiene un QR delante suya lo leerá indefinidamente, esto activará las acciones que le hayamos programadas. Por ejemplo, una vez que lea el QR, el NAO se moverá y hablará, pero si está realizando la acción de hablar y moverse podrá seguir leyendo códigos, si esto ocurre, una vez que acabe de hablar y moverse lo hará una segunda vez y así lo hará tantas veces como haya leído el QR. El robot sabrá cuántas veces ha leído el QR y realizará el código tantas veces como lecturas haya hecho.

Estímulos por defecto: Para que el comportamiento del robot sea lo más interesante para las personas el robot tiene habilitado unos estímulos. Estos estímulos interfieren en el correcto desarrollo del programa pensado y hacen que la tarea de reconocer un código QR sea mucho más compleja.

Global Interpreter Locker en Python: Imposibilidad de ejecutar dos hilos en Python que accedan a memoria al mismo tiempo. El GIL existe para asegurarnos de que no corrompamos la memoria durante la ejecución del programa.

Problemas de conectividad: Este fue un problema recurrente en cuanto al uso del sistema operativo Linux. La red no se mantenía estable, sino que experimentábamos recurrentes cortes en la red, imposibilitando tanto la búsqueda en Internet como la comunicación entre diferentes dispositivos.

4.3 Soluciones

La solución de la doble lectura del código QR fue sencilla. El Nao trabaja mediante suscripciones a eventos. Uno de estos eventos es “BarcodeReader/BarcodeDetected()” que es el que se encarga de detectar si tiene un código QR delante y de obtener la información del mismo.

Por propia definición de estas suscripciones a eventos, el robot estará intentando saber si tiene un QR delante todo el tiempo y en caso de que lo tenga estará leyendo su información todo el tiempo.

Esto nos lleva a un problema, ya que, si tiene programado decir lo que reciba por QR, al dejarle el QR delante unos segundos más, te leerá la información dos veces, en caso de tener la identificación de cliente en forma de QR delante, te leerá esa ID dos veces.

Para resolver ese problema he implementado la solución más sencilla, creé una lista que tiene solo 1 entrada, "Nada", cuando el robot obtenga la ID del cliente la comparará con el primer valor de la lista, si este es el mismo no hace nada, pero si diferente actualizará la lista con ese nombre y lo dirá.

Al iniciar el NAO, este realiza unas funciones, como la de escuchar si se le dice algo, buscar personas o mover la cabeza como si de una persona se tratase. Todo este comportamiento se recoge en ALBasicAwareness. Este módulo permite que el robot pueda ser estimulado por el medio que lo rodea.

Si el robot es estimulado, este intenta buscar su origen y determina si es una persona o no. Si es una persona, el robot pasa al estado "engaged". Sino no encuentra a nadie, el robot seguirá con su tarea anterior, moviendo la cabeza hacia los lados para intentar reconocer a otra persona.

Hay 4 tipos de estímulos:

- Sonido: ALSoundLocalization
- Movimiento: ALMovementDectection
- Personas: ALPeoplePerception
- Tacto: ALTouch

Este conjunto de estímulos han de ser desactivados para una detección más sencilla del QR. Si estuviesen activados, sería muy difícil que el NAO obtuviese la información del QR ya que no deja de mover la cabeza durante el tiempo necesario.

Por ello, en el código existe una función llamada "manejaEstimulos" que recibe 3 argumentos, la ip del robot, el puerto del robot, y true o false según se quieran activar o desactivar los estímulos.

La idea de los hilos en Python era para ejecutar varios nodos de forma paralela, pero estos hilos accedían a memoria al mismo tiempo. Python tiene un mecanismo que no deja ejecutar dos hilos al mismo tiempo si acceden a memoria, ya que si uno de ellos corrompe la memoria, el funcionamiento del programado se verá distorsionado. La solución no tuvo un gran complicación, solo ahonde todo lo que pude en la documentación hasta que encontré una forma de realizar acciones en segundo plano. A partir de aquí me olvide de los hilos en Python y utilicé esta estructura.

Respecto a los problemas de conectividad poco se podía hacer, el problema aparecía cuando usaba Ubuntu. Con Windows no había ningún problema. En la comunicación entre mi ordenador y el NAO usé cable Ethernet para no tener ningún problema. Para el resto de casos solo quedaba tener paciencia y reintentar hasta que la red quería colaborar.

5. Integración del robot NAO en ROS

ROS fue una herramienta novedosa para mí, por ello explicaré un poco de su historia, una breve descripción acerca de los servicios y mensajes y una guía acerca de cómo configurar ROS para poder usarlo con nuestro Nao. Cabe destacar que la documentación existente acerca del uso de ROS con el Nao está desactualizada, esto hace que al seguirla aparezcan numerosos errores. Por ello, he desarrollado una guía sin errores, la cual está al final de este punto.

5.1 Introducción a ROS

ROS, Robot Operating System, es un framework flexible para la creación de software robótico. Es un conjunto de herramientas, librerías y normas para simplificar la compleja tarea de creación de comportamientos a través de una gran variedad de plataformas.

ROS es un gran proyecto con muchos predecesores y colaboradores. La necesidad de un marco de colaboración abierta fue tal en la comunidad de investigación robótica que muchos proyectos fueron enfocados hacia este objetivo.

Ilustración 38 Robot STAIR.

En el 2000 la Universidad de Stanford realizó diversos intentos para integrar una IA (Inteligencia Artifical), como el robot STanford AI Robot (STAIR) y el programa Personal Robots (PR), que crearon prototipos internos de sistemas de software flexibles y dinámicos destinados a la robótica. En 2007, Willow Garage, una idea visionaria de una incubadora robótica, proporcionó significativos recursos para ampliar estos conceptos y crear mejoras implementaciones. El esfuerzo fue impulsado por un gran número de investigadores que contribuyeron con su tiempo y experiencia tanto a las ideas principales de ROS como a sus paquetes de software fundamentales. El software se desarrolló bajo una licencia de código abierto BSD permisiva, y poco a poco se ha convertido en una plataforma ampliamente utilizada en la comunidad de investigación robótica.

Ilustración 39 Willow Garage.

Desde sus inicios, ROS fue desarrollado en múltiples instituciones y para robots diferentes. Aunque hubiese sido mucho más simple para los contribuyentes colocar su código en los mismos servidores, a lo largo de los años, el modelo "federado" ha surgido como una de las grandes fortalezas del ecosistema de ROS. Cualquier grupo puede iniciar su propio repositorio de código ROS en sus propios servidores

y mantener la propiedad y el control total de la misma. Si eligen poner su repositorio a disposición del público, pueden recibir el reconocimiento y el crédito que merecen por sus logros, y beneficiarse de retroalimentación técnica específica y mejoras.

Actualmente ROS consta de decenas de miles de usuarios en todo el mundo trabajando en dominios que van desde proyectos que son un hobby hasta grandes sistemas de automatización industrial.

5.2 Mensajes y Servicios

Message (msg) types: un mensaje es la información que un proceso envía a otros procesos. Ros tiene muchos tipos estándar se mensajes. Las descripciones se almacenan en my_package/ms/MyMessageType.msg

Service (srv) types: las descripciones de los servicios, almacenadas en my_package/srv/MyServiceType.srv, define las estructuras de datos de solicitud y respuesta para los servicios proporcionados por cada proceso en ROS.

5.3 Uso del robot real

Para despertar al robot hay que ejecutar el siguiente comando:

```
roslaunch nao_bringup nao_full.launch nao_ip:=<robot_ip>
roscore_ip:=<roscore_ip>
```

El primer parámetro proporciona la IP del robot, el Segundo, la IP de roscore, es decir la IP donde se está ejecutando el roscore. Esto es necesario ya que no hay ningún roscore en ejecución en el robot. El tercer parámetro, network_interface, específica la interfaz de red que está conectada. De forma predeterminada, se establece en eth0.

Este comando iniciará la configuración predeterminada del robot con los siguientes publicadores (publisher):

- joint_states
- tf
- top camera
- bottom camera
- left sonar
- right sonar
- microphone

También se puede usar PythonSDK, el cual ha tenido que ser instalado previamente y su PYTHONPATH configurada correctamente en las variables de entorno:

```
roslaunch nao_bringup nao_full_py.launch nao_ip:=<robot_ip>
roscore_ip:=<roscore_ip>
```

Una vez que esta ejecución ha finalizado, podemos abrir RVIZ. El paquete bringup proporciona el fichero de configuración de rviz. Simplemente abriendo esta configuración podremos ver el modelo del robot, tf (árbol de transformadas) y la cámara alta.

Para visualizar el NAO en RVIZ, abrimos un nuevo RVIZ. Tener en cuenta de hacer "source" el fichero setup.bash antes.

```
rosrun rviz rviz
```

Podemos utilizar una configuración predeterminada para RVIZ, la cual tiene todo lo básico como tf, modelo del robot, cámaras y sonares. Esta configuración se encuentra dentro de la carpeta catkin.

```
src/nao_robot/nao_description/config/nao.rviz
```

Basado en la política de Aldebaran Robotics, no podremos visualizar las mallas 3D del robot. Si queremos visualizar un modelo 3D completo tendremos que instalar sus "meshes" y aceptar el acuerdo de licencia. Para ello, solo tenemos que ejecutar:

```
sudo apt-get install ros-indigo-nao-meshes
```

Durante la instalación, tendremos que aceptar la licencia y habremos acabado. Solamente tendremos que volver a ejecutar RVIZ y cargar el archivo de configuración de nuevo. Si hemos realizado correctamente todos los pasos tendremos en nuestro monitor algo parecido a esto:

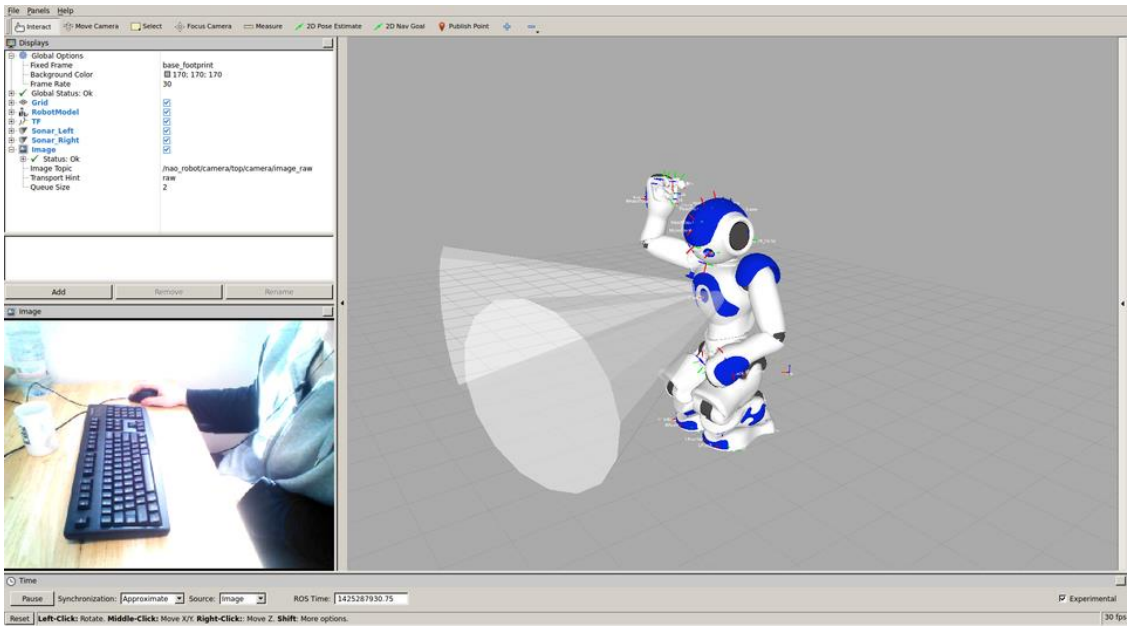


Ilustración 40 Comprobación de la instalación de RVIZ.

Como ejemplo del uso de los nodos del NAO realizaremos un pequeño ejemplo con el nodo walk. Primero debemos comprobar que nodos están ejecutándose, para ello, solo tendremos que listarlos:

```
rosnode list
```

El resultado debería ser un lista con todos los nodos y entre ellos debe aparecer uno que sea /nao_walker.

```
rosnode list

...

/nao_walker

...
```

Es importante, sobre todo en el robot real, tenemos que fijar la rigidez de las articulaciones antes de usar el robot. El siguiente comando dará rigidez a todas las articulaciones y será imposible mover las articulaciones con nuestras manos.

```
rosservice call /body_stiffness/enable "{}"
```

Para deshabilitar esta rigidez en las articulaciones solo tendríamos que introducir el siguiente comando:

```
rosservice call /body_stiffness/disable "{}"
```

Una vez habilitada la rigidez, podemos enviar el comando para caminar:

```
rostopic pub -1 /cmd_vel geometry_msgs/Twist '{linear: {x: 1.0, y: 0.0,
z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}'
```

El robot comenzará a moverse a lo largo del eje X. Además podemos utilizar ROS teleop para controlar el robot con nuestro teclado.

5.4 Uso del robot simulado

El primer paso es ejecutar en un terminal el siguiente comando:

```
roscore
```

A continuación, tenemos que simular el robot, para ello tendremos que ejecutar NAOqi en nuestro ordenador:

```
~/naoqi/naoqi-sdk-1.12.3-linux32/naoqi --verbose --broker-ip 127.0.0.1
```

La dirección del broker-ip es la dirección para conectarse al propio ordenador (localhost). Por lo tanto, solo una instalación ROS local podrá conectarse a NAOqi. El NAOqi API sería accesible a través de la dirección del broker-ip y nao_driver actúa como contenedor para NAOqi API.

A continuación ejecutamos el paquete ros_driver, conectándolo al Nao simulado en nuestro ordenador. Ejecutamos lo siguiente:

```
LD_LIBRARY_PATH=~/naoqi/naoqi-sdk-1.12.3-linux32/lib:$LD_LIBRARY_PATH
NAO_IP=127.0.0.1 roslaunch nao_driver nao_driver_sim.launch
```

Nota: No añadir el path anterior a nuestro LD_LIBRARY_PATH en ~/.bashrc porque hará que otras cosas fallen. Por ejemplo la ejecución de rviz fallará y nos mostrará este error

```
Config file '/home/USERNAME/.rviz/config' could not be opened for
reading.

/opt/ros/fuerte/stacks/visualization/rviz/bin/rviz: symbol lookup
error: /opt/ros/fuerte/stacks/visualization/rviz/lib/librviz.so:
undefined symbol: _ZN9QListData11detach_growEPii
```

El fichero nao_driver_sim.launch establece use_joint_sensors=false y luego llama al fichero nao_driver.launch. Si el anterior parámetro está a true durante la simulación, el topic /joint_states de ROS no se actualizará y no funcionará.

Hay un nao_driver_2.launch que establece use_joint_sensors=false e intenta cargar el paquete nao_footsteps. Si lo conectamos al robot real, tenemos que asegurarnos que la versión de NaoQi instalado en el robot es compatible con nao_driver.

```
If successful, you should see an output similar to:


Walker online...

.

nao_controller running...
```

Para visualizar al Nao en RVIZ primero tenemos que instalar rospack.

```
sudo apt-get install ros-fuerte-rospack
```

Luego instalar las herramientas de visualización para ROS, incluyendo rviz:

```
sudo apt-get install ros-fuerte-visualization
```

Debemos volver a iniciar un roscore, naoqi y nao_driver como en la sesión anterior

Para comprobar la instalación de rviz, ejecutamos:

```
rosrun rviz rviz
```

Si todo esta correcta, una GUI debería cargarse. Si es así podemos cerrar rviz.

Nota: la primera vez que ejecute rviz podría visualizar este error:

```
Config file '/home/USERNAME/.rviz/config' could not be opened for
reading.
```

Una vez que ha guardado el archive de configuración mientras salimos de rviz, no aparece el error otra vez.

Ahora necesitamos iniciar el publicador robot_state_publisher y este cargará el modelo URDF del robot:

```
roslaunch nao_description nao_state_publisher.launch
```

Luego, volvemos a ejecutar rviz.

```
rosrun rviz rviz
```

En la esquina superior izquierda se encuentra la ventana “Displays”, tenemos que cambiar “Fixed Frame” a “/base_link”. Si solo la opción “/map” está disponible, entonces el modelo URDF no ha sido cargado del paso anterior.

El “Target Frame” debe ser “<Fixed Frame>”.Global Status debería cambiar a “OK”.

Si está en rojo y aparece “error” entonces significa que el topic /joint_states no está siendo actualizado.

Hacemos clic en “Add” y añadimos grid y RobotModel.

Si todo está bien, se verá el modelo del robot hecho de cilindros y el robot se encontrará en la posición de descanso.

Ahora publicaremos la orden para que el robot comience a andar en rviz:

```
rostopic pub -1 /cmd_vel geometry_msgs/Twist '{linear: {x: 1.0, y: 0.0,
z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}'
```

Detenemos el robot:

```
rostopic pub -1 /cmd_vel geometry_msgs/Twist '{linear: {x: 0.0, y: 0.0,
z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}'
```

El modelo del robot esta descrito por la estructura del archive .xacro. Para crear un modelo mejor, tendremos que editar el archive visuals.xacro que se encuentra en el directorio:

```
~/ros_workspace/nao/stacks/nao_common/nao_description/urdf/
```

5.5 Mi experiencia.

Debido al coste inicial del Nao, hay muy pocas actualizaciones, por no decir que ya no están interesados en seguir actualizándolo. Por lo tanto, algunas veces se producen fallos o los directorios tienen otros nombres, tanto fue así que a la tercera vez que experimenté estos errores decidí realizar mi propia guía de instalación.

Una vez descargados los archivos necesarios, pasaremos a descomprimirlos. Los archivos de los que hablo son naoqi-sdk-2.1.4.13-linux64.tar y pynaoqi-python-2.1.4.13-linux64.tar (descargados de la página web de Aldebaran Robotics). A continuación, podemos ejecutar en un terminal lo siguiente:

```
~/naoqi/naoqi-sdk-2.1.4.13-linux64/naoqi
```

Añadimos lo siguiente a las variables de entorno, bien sea abriendo nuestro bashrc y escribiéndolo al final o ejecutando lo siguiente en un terminal:

```
echo 'export PYTHONPATH=~/naoqi/pynaoqi-python2.7-2.1.4.13-
linux64:$PYTHONPATH' >> ~/.bashrc
```

Ejecutamos Python y probamos a introducir:

```
from naoqi import ALProxy
```

Si no salta ningún mensaje dentro de Python es que estamos en el camino correcto.

Lo siguiente es instalar todos los paquetes que necesitemos. Este punto es muy tedioso. La explicación es muy sencilla, hay paquetes que dependen de otros pero eso no podemos saberlo a simple vista. Al intentar ejecutar el paquete pueden pasar dos cosas, que se ejecute sin problemas, eso quiere decir que no depende de ningún otro y podemos pasar al siguiente, o bien que salte un error y en ese mismo mensaje nos indique que paquete necesita. Otra forma de ver esas dependencias sin necesidad de ejecutar nada es dirigirnos al CMakeList.txt del paquete y ahí encontraremos todas sus dependencias, solo tendremos asegurar de tener instalados todos los paquetes que necesite antes de ejecutarlo. Para encontrar los paquetes referentes al robot solo hay que escribir lo siguiente:

```
sudo apt-get install ros-indigo-nao (y pulsar 2 veces el tabulador)
```

Aun así, si se ha instalado ROS por primera vez es muy posible que falten algunos paquetes referentes a ROS en general, para buscarlos habría que realizar algo parecido a la línea de código de arriba:

```
sudo apt-get install ros-indigo (y pulsar 2 veces tabulador)
```

Una vez instalados todos los paquetes, ejecutamos un roscore, que es la herramienta principal de ROS, sin los nodos no pueden comunicarse entre sí.

```
roscore
```

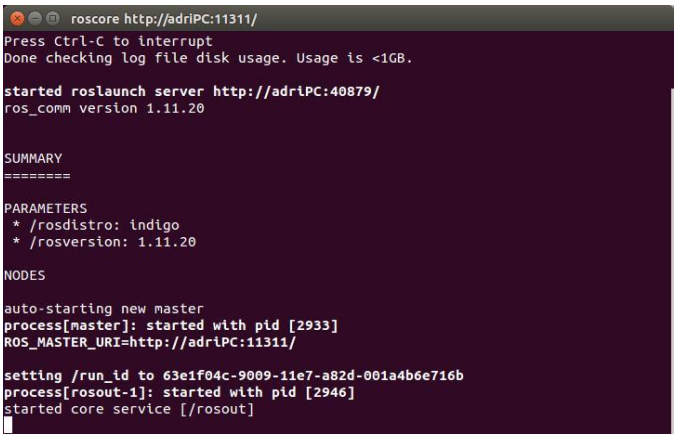


Ilustración 41 roscore.

En otro terminal:

```
~/naoqi/naoqi-sdk-2.1.4.13-linux64/naoqi
```

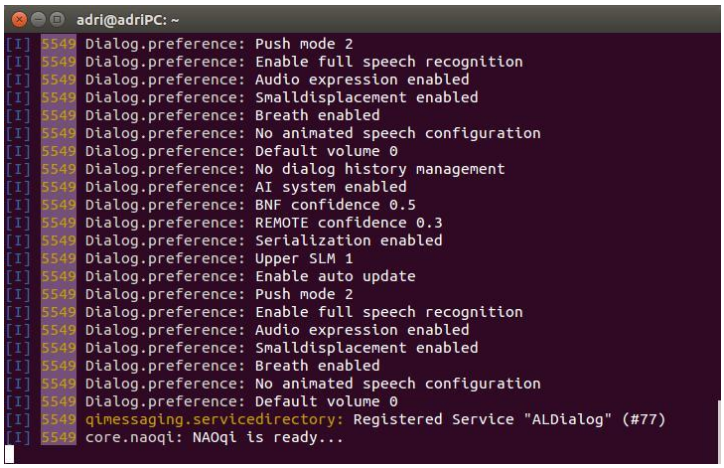


Ilustración 42 NAOqi preparado.

En otro aparte:

```
roslaunch nao_bringup nao_full_py.launch
```

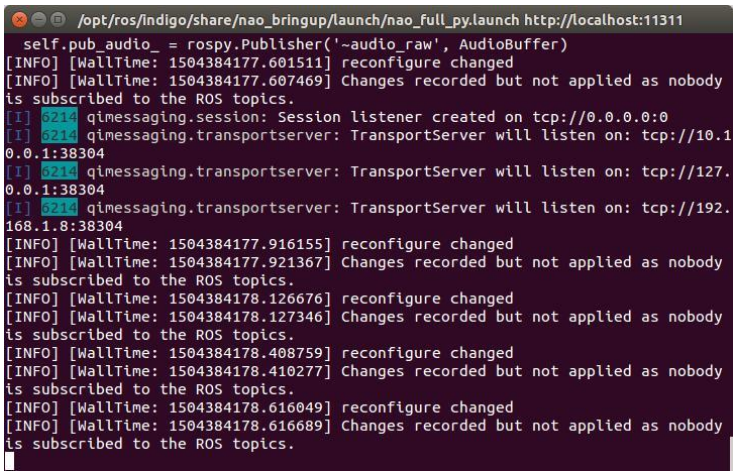


Ilustración 43 nao_bringup.

Finalmente ejecutamos el nodo publicador del estado del robot y el rviz para visualizar todo (terminales diferentes):

```
roslaunch nao_description robot_state_publisher.launch

rosrun rviz rviz
```

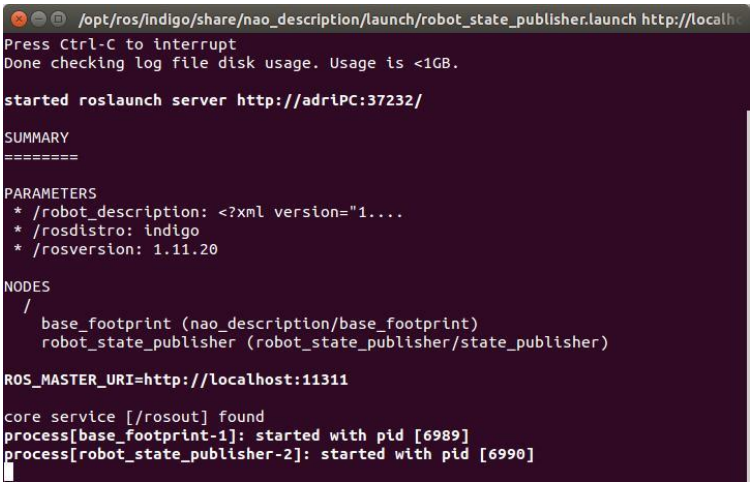


Ilustración 44 nao_description.

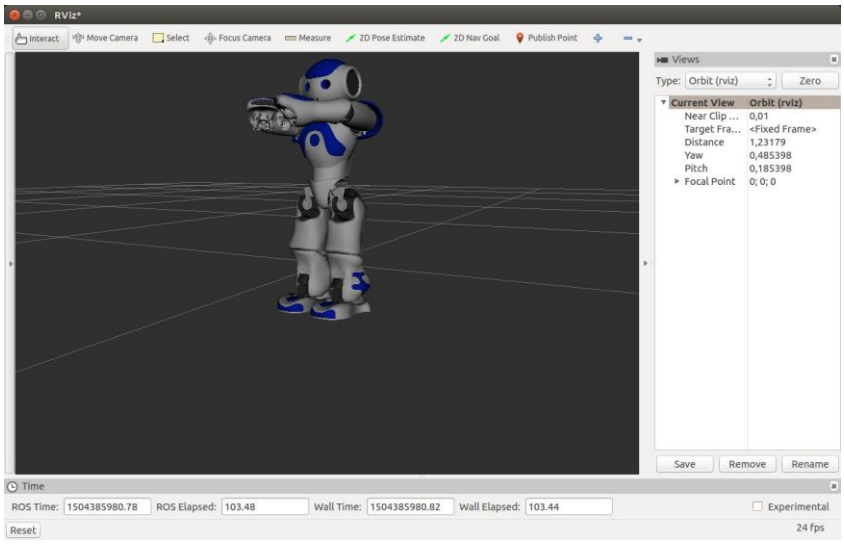


Ilustración 45 Nao en rviz.

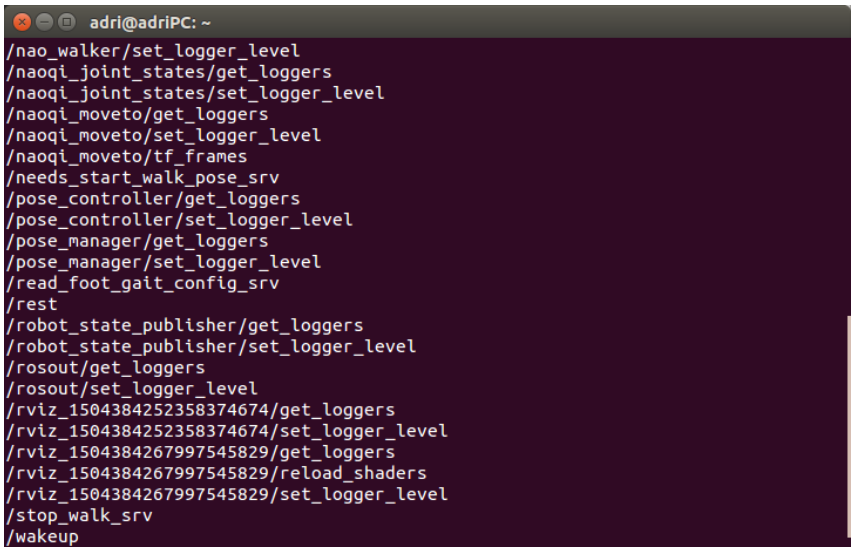


Ilustración 46 rosservice.

La imagen superior muestra los servicios que podemos llamar para realizar acciones con el Nao y visualizarlas en rviz. Lamentablemente, entre diferentes errores que me encontré a lo largo del proceso y que aparecen al actualizar algunos paquetes y que es necesario saber mucho más de ROS para sacar el verdadero partido a este robot en este entorno solo pude desarrollar pequeños programas para crear servicios y hacer pequeñas acciones como la siguiente.

```
1  #include "ros/ros.h"
2  #include "nao_programs/points.h"
3
4  int main(int argc, char **argv)
5  {
6    ros::init(argc, argv, "nao_anda_server");
7    ros::NodeHandle n;
8    ros::ServiceServer service = n.advertiseService("nao_anda", add);
9    ROS_INFO("El robot esta listo para caminar.");
10   ros::spin();
11
12   return 0;
13 }
```

Ilustración 47 Breve ejemplo de programa.

6. Conclusión

Pasamos a explicar que he conseguido en este proyecto, tanto en lo profesional como en lo académico, divido en el resultado del proyecto, es decir, como finalizó el proyecto y como el trabajo transformado mi forma de pensar.

6.1 Resultado del Proyecto

Los resultados dentro del proyecto RETAIL han sido los siguientes:

- Nao es capaz de reconocer códigos QR y de procesar la información.
- El servidor del lado del robot envía la id obtenida por el robot del código que el cliente le ha mostrado al servidor del lado de la empresa o compañía.
- El robot puede transmitir al cliente ofertas recientes basadas en la información en la base de datos de la empresa.
- El robot gesticula al hablar para atraer la atención de los clientes.

Resultados aparte del proyecto, centrados en el aprendizaje propio han sido:

- Desarrollo de habilidades de programación en diferentes lenguajes (JAVA, Python y C++).
- Aprendizaje acerca de ROS y POO (Programación Orientada a Objetos).
- Conocimientos sobre la comunicación a través de sockets.
- Mejora de habilidades:
    i) Búsqueda de información.
    ii) Enfoque de problemas.
    iii) Desarrollo de documentación.

Los 4 primeros puntos fue lo que la empresa, Tier 1, nos pidió como objetivo. Obviamente, el robot puede hacer infinidad de cosas, pero ellos querían que lo que hiciese fuese robusto de cara a próximas presentaciones. Aun así, por cuenta propia he llegado a desarrollar programas para que el robot pueda mantener una simple conversación, realice “face-tracking”, camine evitando obstáculos, tanto en Choregraphe, en Python o ROS.

Por ello, desde fuera el objetivo del robot puede parecer simple, aunque si empiezas a programarlo se puede ver que hay que manejar métodos y conceptos con los que no estaba familiarizado y los cuales necesitan de una inversión en tiempo para poder dominarlos y llegar a realizar el objetivo que se tenía en mente.

Con este proyecto podemos tener en cuenta los posibles caminos a elegir dentro del desarrollo de aplicaciones muy interesantes dentro de diversos campos. Tanto es así que desde que Aldebaran Robotics creó su primer NAO en 2006, se han vendido más de 9000 ejemplares a lo largo del mundo, destinados a diferentes aplicaciones:

- **Educación:** Desde su aparición el robot NAO cautivó la atención de profesores y estudiantes a lo largo de muchas universidades. Por un lado, es una increíble herramienta de aprendizaje, puede ayudar a los estudiantes acerca de programación o a los profesores a mantener la atención de sus estudiantes.

  Además, NAO puede suponer un gran avance para aquellos estudiantes de robótica que estén empezando con la programación. NAO está repleto de módulos tales como el seguimiento de caras, los cuales despiertan mucho interés entre los estudiantes.

- **Negocios:** Los robots están enfocados a enriquecer y transformar la interacción realizada por el cliente. De esta manera, los autómatas son usados para darles la bienvenida o para mostrarles nuevos productos y servicios. Existen ejemplos de la integración de estos robots en diversas compañías:

  - SoftBank, banco japonés, cuenta con más de 2000 robots Pepper cuyo objetivo es generar interés, anunciando ofertas y saludando a los clientes.

  - Nestlé Japan, la rama nipona de la famosa empresa de alimentación cuenta desde 2014 con 1000 Peppers distribuidos en sus diferentes tiendas para hacer más atractivo el producto al cliente.

  - SNCF, Sociéte Nationale des Chemins de fer Français, empresa estatal francesa que se encarga de la explotación de los ferrocarriles, eligió al Pepper para ofrecer una nueva experiencia a los clientes en 3 estaciones de Pays de la Loire. Sus funciones eran la detección e interacción de personas, informar acerca de los tener y de los alrededores, recabar la satisfacción de los clientes y divertir a los clientes que esperaban para las compras de sus tickets.

  - Carrefour también ha tomado la decisión de confiar la tarea de atraer a los clientes mediantes el uso de los Pepper.

Ilustración 48 Uso del robot Pepper en Carrefour.

Actualmente la empresa francesa está comercializando su robot Pepper y está desarrollando un nuevo proyecto con el nombre de Romeo. Con toda esta información, podemos tener en cuenta que la robótica experimentará grandes avances en los siguientes años, ver la proyección de los diferentes proyectos que se van a desarrollar y, finalmente, entender que todo el trabajo desarrollado en el proyecto RETAIL ha tenido un enfoque más que adecuado.

6.2 ¿Qué ha supuesto el TFG como persona/ingeniero?

Cuando el proyecto comenzó yo no conocía ninguna de las herramientas que posteriormente utilicé, así como Linux, si bien es cierto que hemos usado este sistema operativo también cabe decir que no lo hemos visto en profundidad. Por ello, las primeras semanas del proyecto estuve leyendo páginas y páginas en busca de la manera de empezar con todo lo que tenía por delante.

No fue nada fácil, ya que estaba acostumbrado a que todo nos fuese dado o nos costase un trabajo mínimo encontrarlo. Tanto fue así, que mucha información no fue útil al principio pero sirvió para ir cerciorándome de la profundidad del robot.

Aun así, todo el trabajo de búsqueda de información fue bastante positivo para mí. Como ya he dicho, salvo en contadas asignaturas, no he encontrado demasiados problemas para realizar cualquier programa o proyecto y son esos problemas los que realmente te ayudan a crecer.

Choregraphe fue el programa que usé en primer lugar, ya que su dificultad es mínima. A continuación, le siguió Python y, por último, utilicé Python y C++ en el entorno de ROS. Todo esto estuvo acompañado por un aprendizaje en Linux, como configuración del bash, sockets y actualizaciones.

Por otro lado, también he programado en el lenguaje de programación JAVA, para la creación de los servidores. La profundidad en este lenguaje de programación ha sido demasiado superficial. Personalmente, quise realizar un aprendizaje más profundo acerca de este lenguaje y así equipararlo con el nivel de Python y C++. De todas maneras, lo único que he visto en la carrera respecto a JAVA han sido un par de prácticas en la asignatura de Arquitectura de Redes y el código nos venía dado, así que, esta ha sido la primera vez que he tenido que realizar el código desde 0. De esta forma experimenté las diferencias entre los 3 lenguajes y ver cómo están estructurados hace que puedas elaborar nuevas soluciones ya que cambian tu modo de pensar.

Por ello, el resultado personal ha sido increíble. Durante el TFG solo se me proporcionaban problemas a resolver pero no me exigían resultados inmediatos. Era yo quien quería la mejor solución posible, por ello he tenido que buscar mucha información respecto a todos los ámbitos relacionados. Con ello he desarrollado con creces la habilidad de búsqueda de información relevante, desechando toda aquella que no es interesante.

Por último, este proyecto ha sido el aliciente que necesitaba. Estaba cansado de tener que resolver problemas teóricos o programas que un objetivo simple, esto hizo que mi interés en el grado descendiese enormemente. Pero al encarar este proyecto me he dado cuenta que todo lo enseñado en las diferentes asignaturas del grado tienen su objetivo. Ya sea proporcionarte los conocimientos para resolver problemas o para ampliar tus puntos de vista.

7. Anexos

En este punto encontraremos la información necesaria para poder ejecutar todo lo realizado en el proyecto, Manual de Usuario, como una descripción de los programas más importantes de los que se han desarrollado.

7.1 Manual de Usuario

7.1.1 Instalación

7.1.1.1 Instalación Choregraphe

7.1.1.1.1 Windows

Lo primero que tenemos que hacer para proceder a la instalación del software de Aldebaran Robotics en dirigirnos a su página y descargárnoslo, para ello previamente debemos realizar el registro en su página web. Para ello podemos hacer clic en "Sign in" y luego en "Create an account" o copiar y pegar el link en la barra de direcciones del navegador (https://store.aldebaran.com/default/customer/account/create/).

Lo siguiente para la creación de la cuenta es cumplimentar la siguiente información.

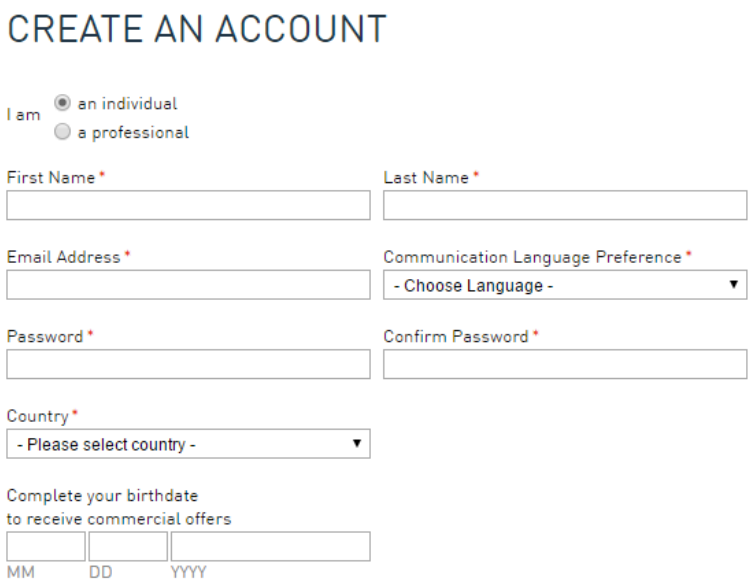


Ilustración 49 Creación de una cuenta en Aldebaran Robotics.

Para acceder a la lista de todos los instaladores disponibles tenemos que hacer clic en Community > Resources > Software o usar el siguiente link https://community.ald.softbankrobotics.com/en/resources/software/language/en-gb

Para descargar el instalador adecuado debemos buscar el acorde a nuestro sistema operativo de entre todos los disponibles.

Ilustración 50 Diferentes posibilidades de descarga del instalador.

Para descargarlo no tenemos más que hacer clic sobre el deseado y esperar el tiempo de descarga. Una vez terminada la descarga, presionamos sobre el instalador y se nos abrirá la siguiente ventana:

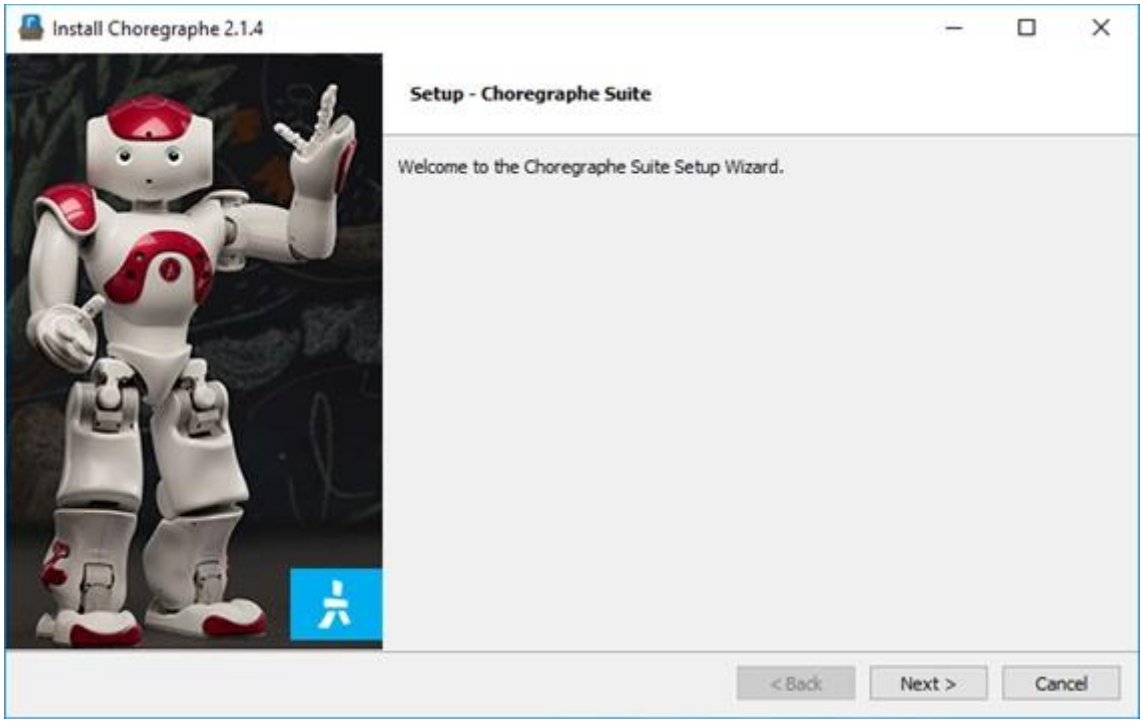


Ilustración 51 Comienzo del proceso de instalación.

Solo tenemos que hacer clic en “Next” y nos aparecerá el acuerdo de licencia del software, en esta parte debemos leerlo, aceptarlo y pulsar “Next”.

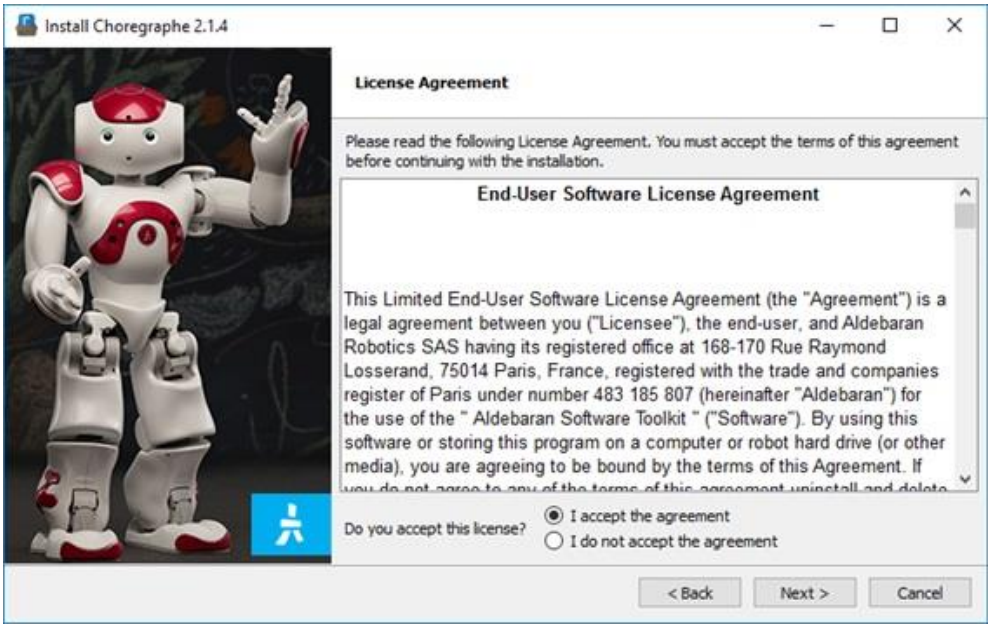


Ilustración 52 Aceptación del contrato de licencia del software.

A continuación, el software nos pide una clave de licencia, en mi caso no tenía ninguna, así que seleccione la segunda opción que te permite probar el programa por un periodo de 90 días. Aunque si se dispone de una clave se puede elegir la primera opción, teniendo así el programa de por vida.

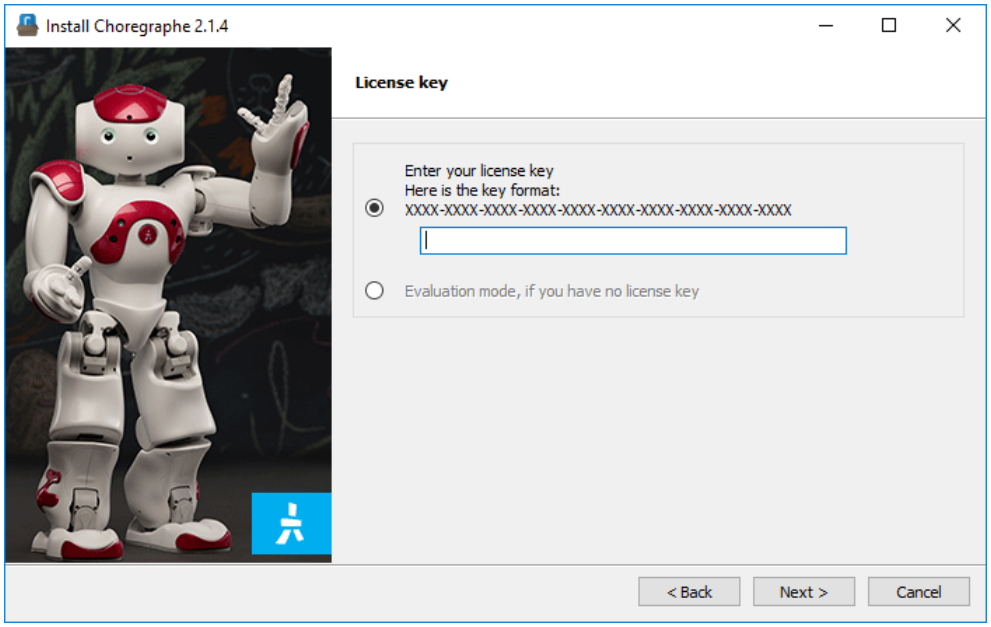


Ilustración 53 Inserción de la clave de licencia.

El siguiente paso es seleccionar el tipo de instalación que deseamos, “Quick” significa que instalará de manera automática y “Advanced” es para usuarios con más nociones que quieran hacer una serie de personalización de la instalación. Para pasar al siguiente paso hacemos clic en “Next”.

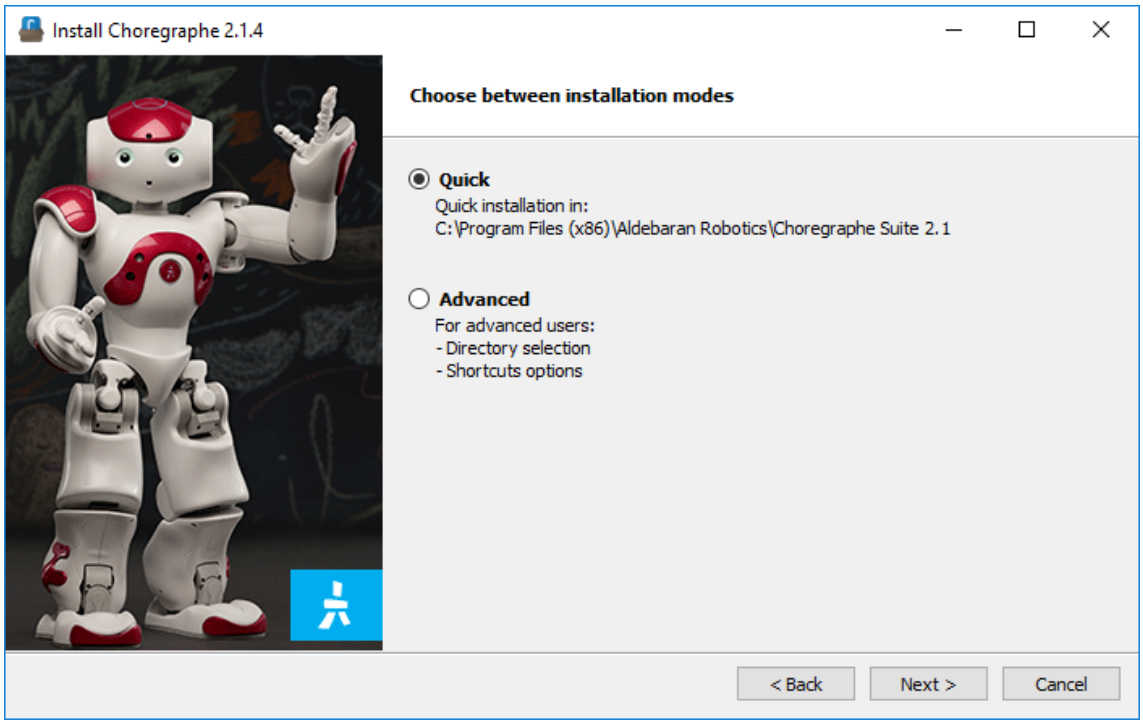


Ilustración 54 Elección del modo de instalación.

Cuando empiece la instalación veremos una ventana como la siguiente:

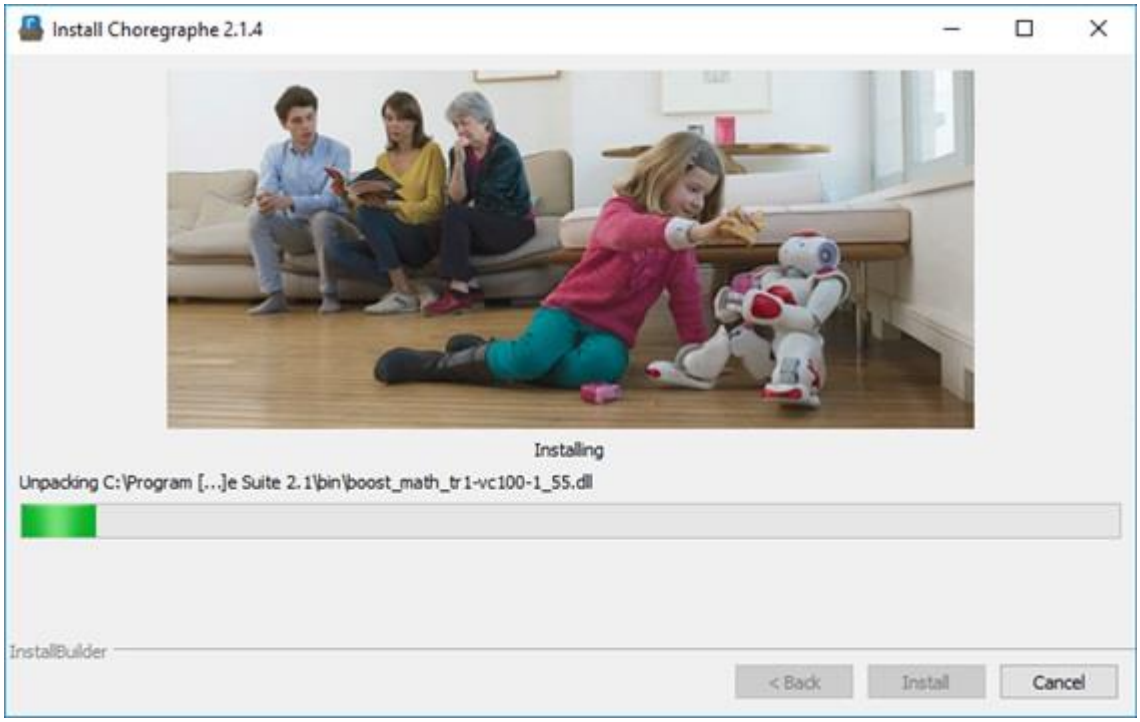


Ilustración 55 Instalación en curso.

Cuando la instalación ha finalizado se nos mostrará la siguiente pantalla, pudiendo seleccionar/deseleccionar la opción "Launch Choregraphe 2.1.4" para la ejecución o no automática del programa justo después de todo el proceso. Para finalizar tenemos que hacer clic en "Finish".

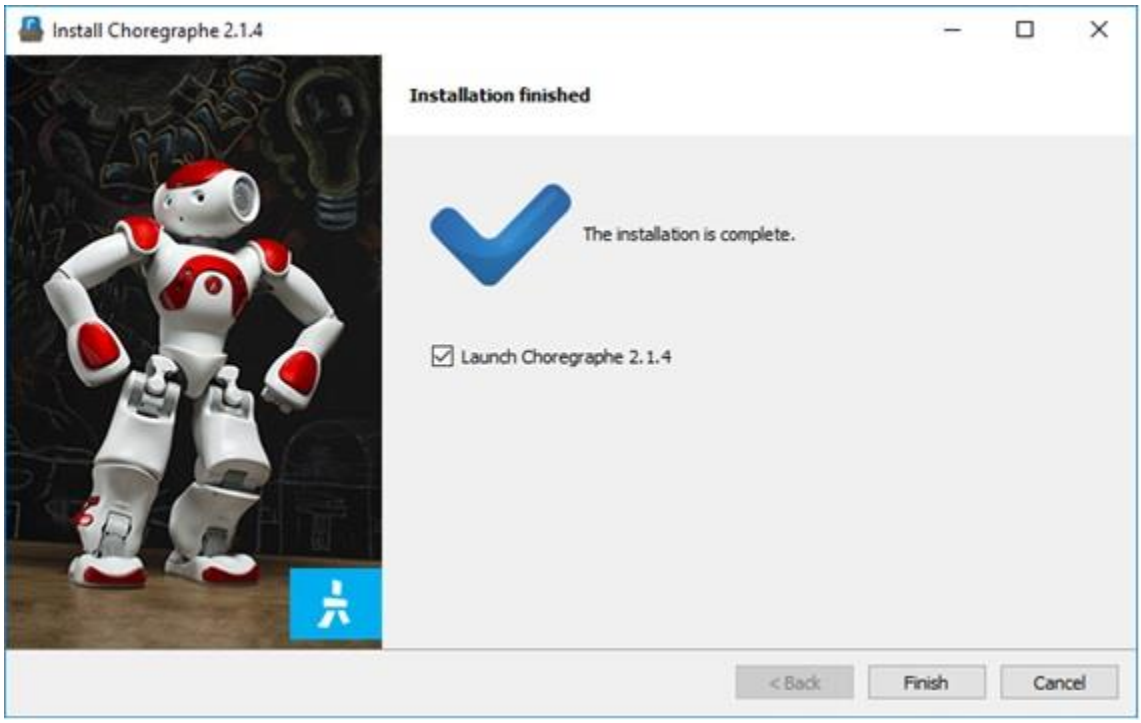


Ilustración 56 Instalación finalizada.

Cada vez que ejecutemos el programa veremos la siguiente pantalla:

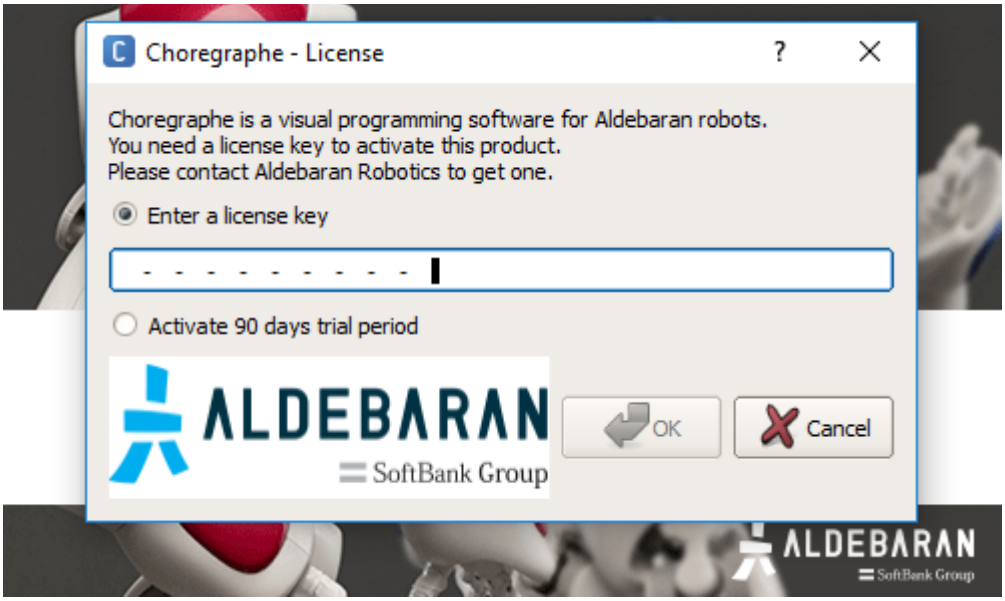


Ilustración 57 Primera ejecución del programa.

El software nos pide la licencia o activar el período de 90 días. En este paso solo hay que seleccionar la opción deseada, completar la información si procede, y pulsar “Ok”.

7.1.1.1.2 Ubuntu

Aunque Aldebaran Robotics nos proporciona una guía de instalación (http://doc.aldebaran.com/2-1/getting_started/installing.html#desktop-installation), experimente bastantes problemas.

Lo que hice fue darle permisos de ejecución al archivo. Para ello, tendremos que abrir un terminal y dirigirnos donde se encuentre el fichero en cuestión, a continuación, solo hay que escribir "chmod +x choregraphe-suite-2.1.4.13-linux32-setup.run" luego solo hay que hacer doble clic al archivo. Se iniciará un proceso de instalación (mismo proceso que el explicado anteriormente en el apartado 3.1.1.1) en el que solo importa el momento de introducir la licencia. Al no disponer de una licencia se eligió la opción de la prueba de 90 días.

7.1.1.2 Instalación PythonSDK

7.1.1.2.1 Windows

SDK son las siglas de "Software Development Kit" o "Kit de Desarrollo Software", es decir, este paquete de Python nos permite trabajar en el entorno del robot, pero para ello tenemos que tener Python instalado. Además, necesitamos instalar la versión correcta. Para saber cuál necesitamos solo tenemos que fijarnos en el nombre del archivo a descargar, como podemos ver en la siguiente imagen:

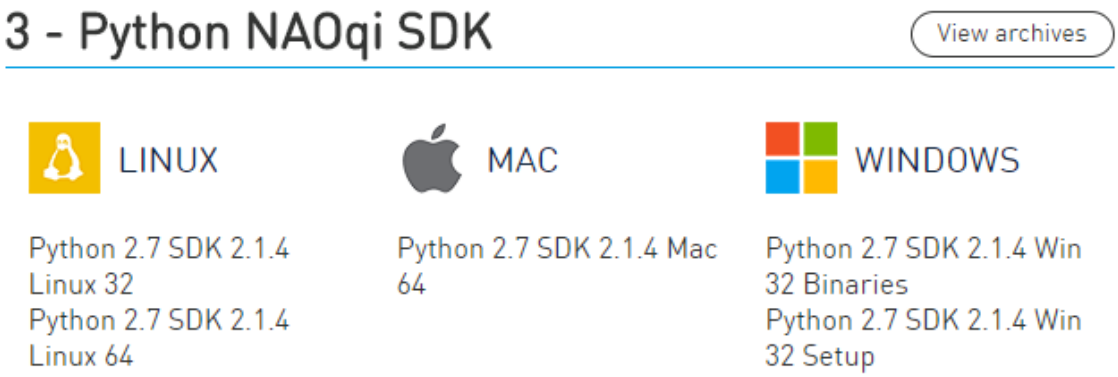


Ilustración 58 Diferentes instaladores de Python NAOqi SDK.

La versión que buscamos es la 2.7. A continuación, nos dirigimos a su página web para proceder a la descarga (https://www.python.org/), hacemos clic en "Downloads". La siguiente página que cargará nuestro navegador mostrará en uno de los dos recuadros amarillos "Download Python 2.7.X", (la X es por diferentes versiones, no importa que número sustituya a la X siempre que este precedido por 2.7). Hacemos clic en ese recuadro, no descargamos el instalador y hacemos clic en él y nos aparecerá la siguiente ventana:

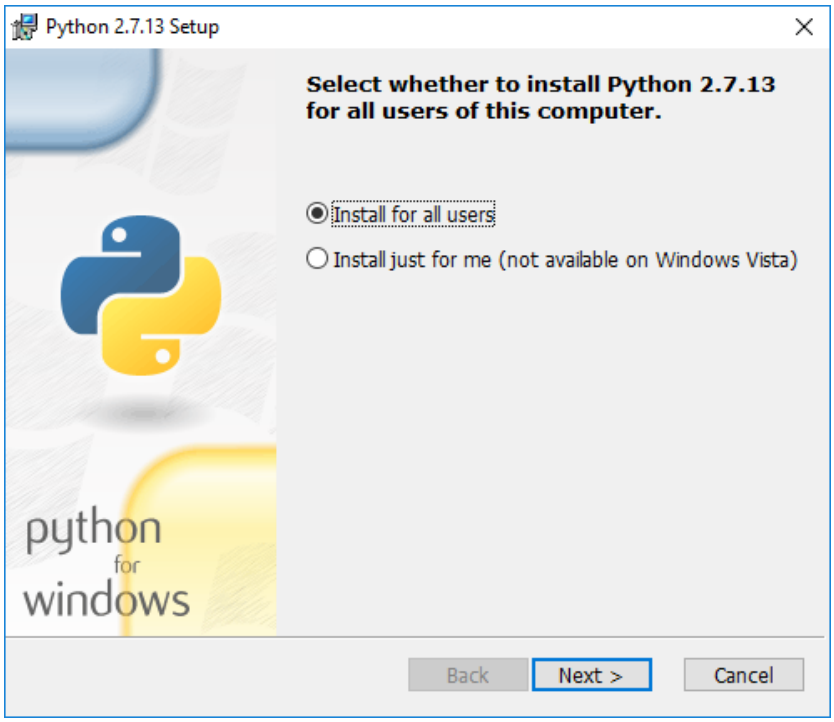


Ilustración 59 Instalación de Python.

El proceso de instalación es bastante sencillo, podemos elegir una de las dos opciones, según queramos Python para todos los usuarios o solo para el actual, a continuación, pulsamos "Next".

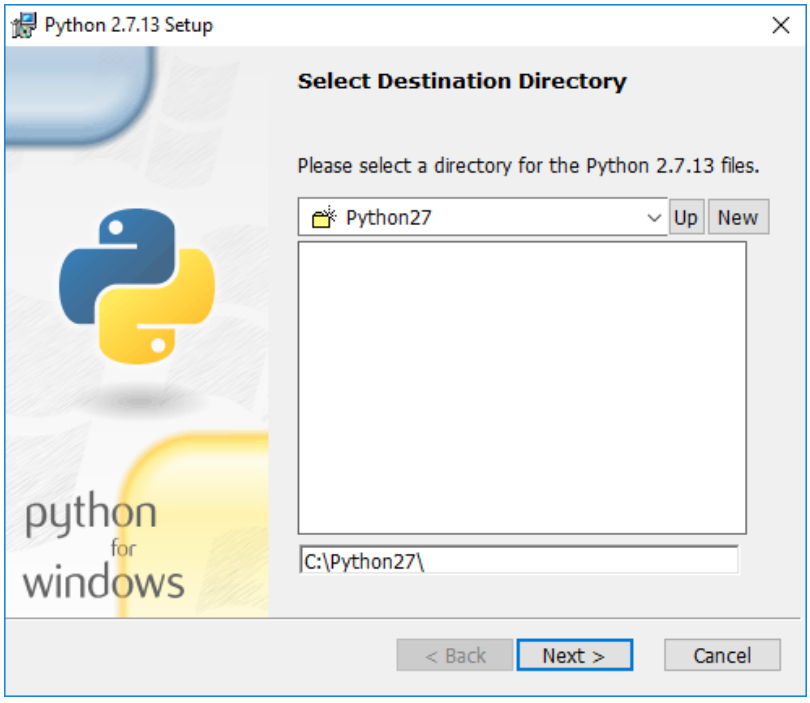


Ilustración 60 Selección de la ruta para la creación del directorio del programa.

Tenemos que seleccionar donde se creará la carpeta de Python, recomiendo dejarla por defecto, y presionar "Next".

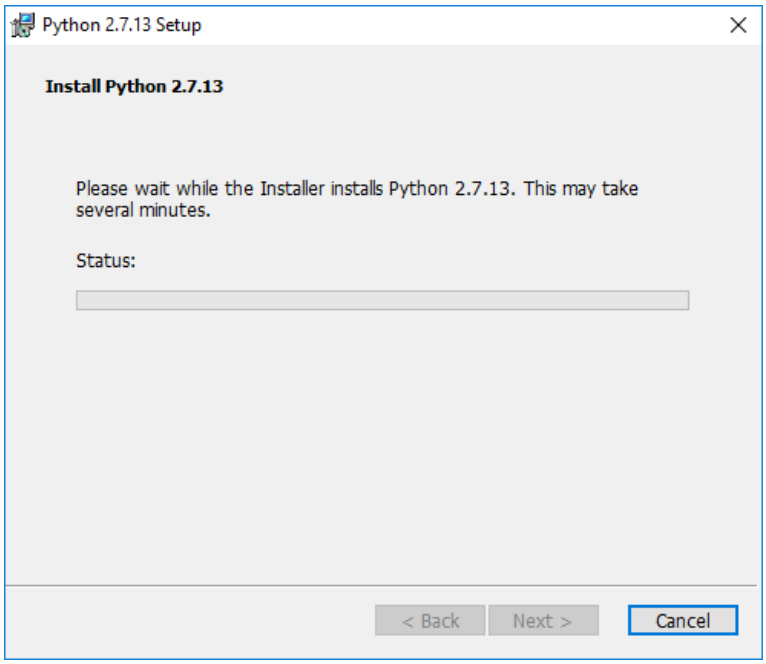


Ilustración 61 Comienza el proceso de instalación.

Al finalizar el proceso de instalación, solo tenemos que hacer clic en “Finish”.

Ilustración 62 Instalación completada.

El ejecutable de Python lo podremos encontrar donde creamos la carpeta, si no se cambió la ruta, debemos irnos a Disco Local (C:) y buscamos la carpeta Python27, entramos en ella y encontramos el ejecutable bajo el nombre “python.exe”. Al hacer clic sobre él, nos aparece la siguiente ventana:

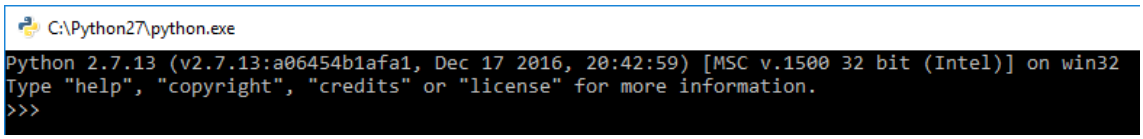


Ilustración 63 Python en Windows.

Hasta ahora hemos instalado Python, ahora tenemos que instalar el framework que nos habilite trabajar con el robot, para ello nos dirigimos a la lista de todos los instaladores (ya comentado en el apartado de Choregraphe o usamos el link https://community.ald.softbankrobotics.com/en/resources/software/language/en-gb en nuestro navegador). Como ya vimos en la ilustración 46, descargamos el instalador de PythonSDK, el que especifica setup, Python2.7 SDK 2.1.4 Win 32 setup, los otros son ficheros que debemos de compilar y son para usuarios avanzados. Haciendo clic en el instalador deseado, pasaremos a su descarga y luego haremos clic sobre él apareciéndonos la siguiente ventana:

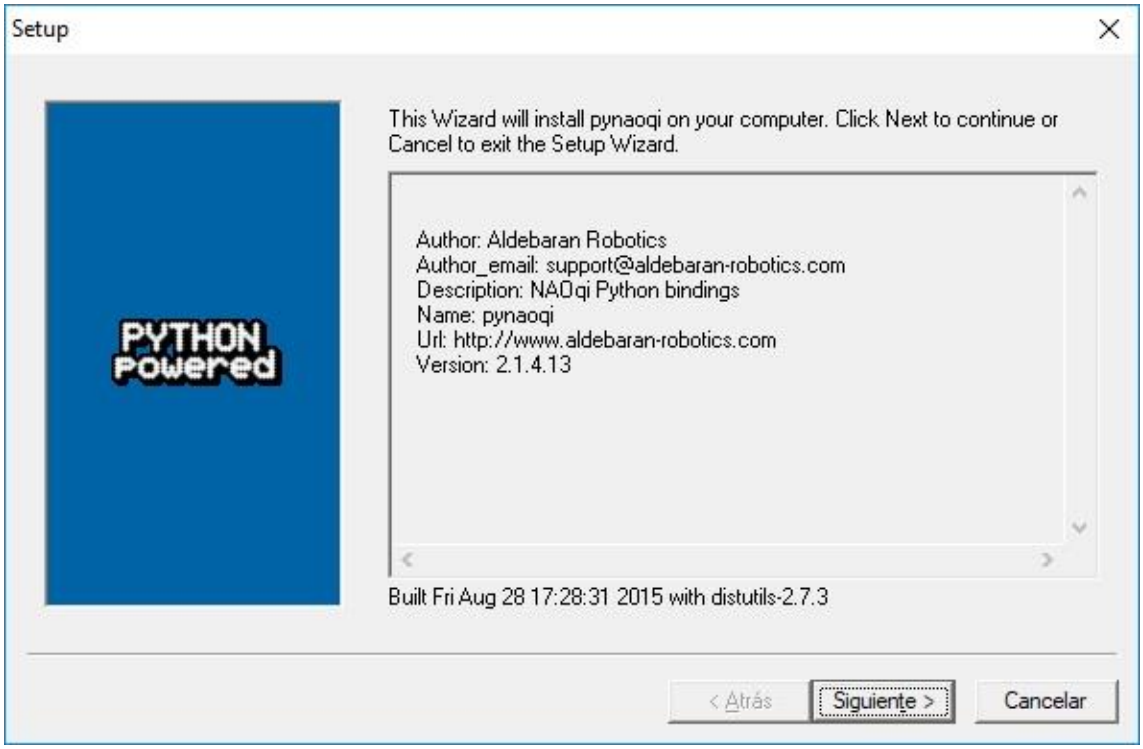


Ilustración 64 Comienzo de la instalación de Python NAOqi SDK.

Esta es una instalación bastante sencilla, solo tenemos que hacer clic en “Siguiente”.

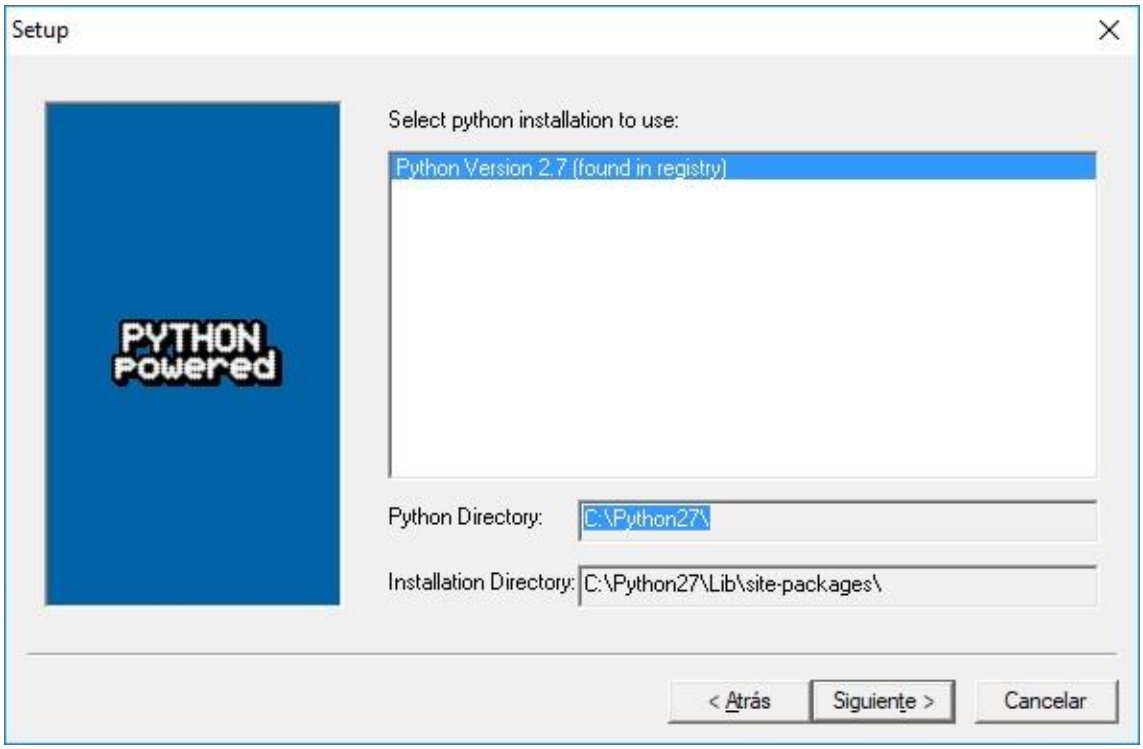


Ilustración 65 Selección del directorio de Python 2.7.

En este paso debemos indicar donde se encuentra la carpeta de Python, si no hemos modificado nada debe encontrarse en C:\Python27\. Si es así, hacemos clic en "Siguiente", si no debemos especificarla debidamente.

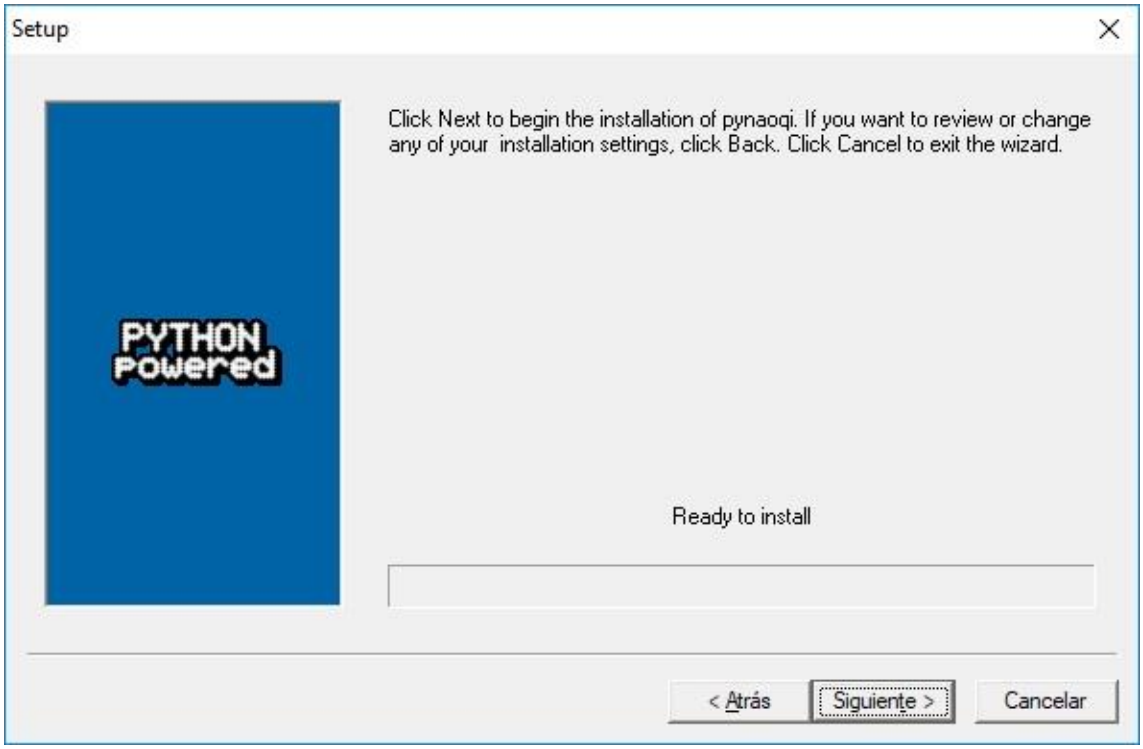


Ilustración 66 Instalación.

Al hacer clic en "Siguiente", comenzará la instalación del SDK.

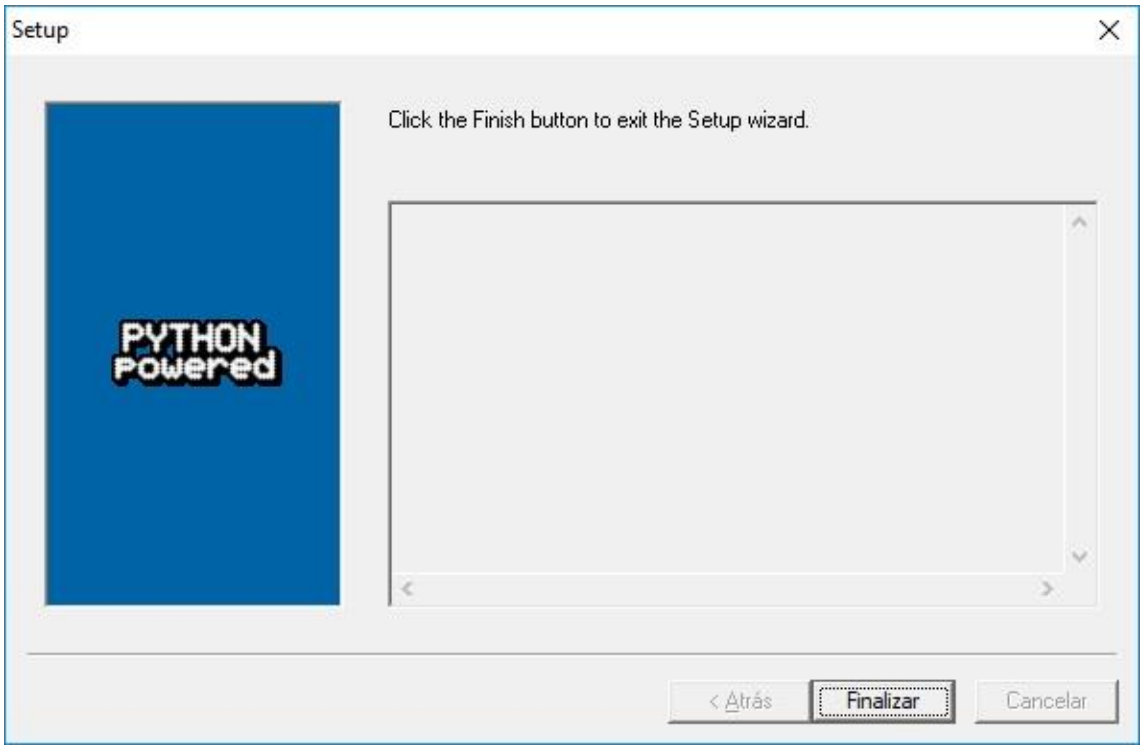


Ilustración 67 Instalación finalizada.

Una vez finalizado el proceso de instalación, solo tenemos que hacer clic en “Finalizar”. Aunque no hemos acabado, queda 1 paso más. Abrimos Python y escribimos “import naoqi” si nos sale igual que en la imagen de abajo, todo el proceso de instalación se ha realizado correctamente.

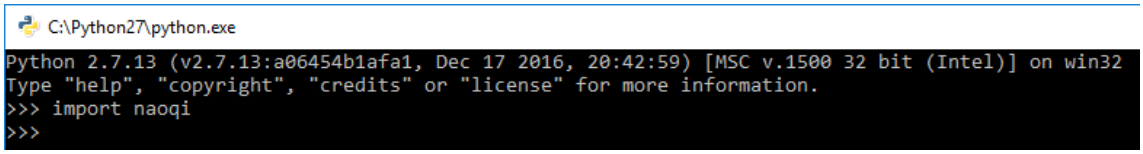


Ilustración 68 Comprobación de la instalación.

7.1.1.2.2 Ubuntu

En la versión utilizada de Ubuntu, la 14.04, Python está incluido, tanto es así que podemos comprobarlo escribiendo:

```
python -V
```

Con ese comando, veremos que versión de Python tenemos instalada, normalmente será la 2.7, sino tendremos que instalarlo y para ello solo tenemos que dirigirnos nuevamente a la terminal de Ubuntu y escribir:

```
sudo apt-get install python2.7
```

Una vez finalizado el proceso de instalación en la misma terminal podemos escribir python para entrar al programa. A continuación, debemos instalar el mismo SDK que instalamos previamente en Windows, para ello nos dirigimos a la lista de todos los instaladores de Python SDK y seleccionamos el setup pero esta vez del sistema operativo Linux.

Una vez acabe la descarga, abrimos una terminal, los ficheros descargados estarán en la carpeta Descargas si no especificamos una ruta diferente en el proceso de descarga. En la terminal de Ubuntu escribimos "cd /Descargas", una vez en la carpeta Descargas, escribimos:

```
ls
```

Con esto veremos todos los ficheros que contiene, uno de ellos tiene que tener el siguiente nombre "pynaoqi-python-2.7-naoqi-2.1.4.13-linux64.tar.gz", tendremos que descomprimir el fichero, para ello escribimos en la terminal:

```
tar xfz pynaoqi-python-2.7-naoqi-2.1.4.13-linux64.tar.gz
```

Cuando finalice tendremos una carpeta con el mismo nombre que el fichero que hemos descomprimido, en dicha carpeta es en la que vamos a trabajar.

Por último, tenemos que añadir una variable de entorno para no tener problemas. Abrimos un terminal y escribimos:

```
nano /etc/environment (si da un error por la orden nano solo hay que instalar
ese editor de texto, sudo apt-get install nano)
```

y una vez dentro del fichero escribimos **PYTHONPATH=${PYTHONPATH}:/pynaoqi-python2.7-2.1.4.13-linux64**. En el caso de que queramos ponerlo en otras carpetas se haría de la misma manera.

Para probar que todo está correcto, abrimos un terminal nos vamos a la carpeta de **pynaoqi-python2.7-2.1.4.13-linux64** y escribimos **python**. Dentro de la shell de Python probamos **import naoqi**, si no da error toda la instalación ha sido realizada correctamente y solo quedaría programar cualquier aplicación que queramos.

7.1.1.3 Instalación C++

7.1.1.3.1 Windows

Primero tenemos que comprobar que tenemos instalado Visual Studio 2010, el cual usaremos como compilador e IDE.

Por otro lado tenemos que instalar qiBuild, una herramienta diseñada para generar proyectos multiplataforma usando CMake. Previa a la instalación de qiBuild debemos instalar la versión 2.8.3 de CMake o superior, el instalador lo obtendremos de aquí http://www.cmake.org/cmake/resources/software.html. Por otro lado, tendremos que instalar Python 2.7 (descrito en la sección Instalación PythonSDK).

Para instalar la herramienta pip:

```
python get-pip.py
```

Usaremos pip para instalar qiBuild:

```
pip install qibuild
```

Debemos añadir `C:\Python27` y `c:\Python27\Scripts` a nuestro PATH.

Una vez realizados estos pasos, debemos configurar qiBuild. Para ello ejecutar el siguiente comando:

```
qibuild config --wizard
```

En este momento tendremos que especificar el path de CMake, el generador de CMake y el IDE que usaremos. De todas formas podremos modificar la configuración en cualquier momento que queramos solo con volver a ejecutar el comando anterior.

Hemos acabado con todos los requisitos previos, ahora nos centraremos en la instalación de C++ SDK. Lo primero que debemos hacer es seleccionar los paquetes adecuados para nuestro sistema operativo.

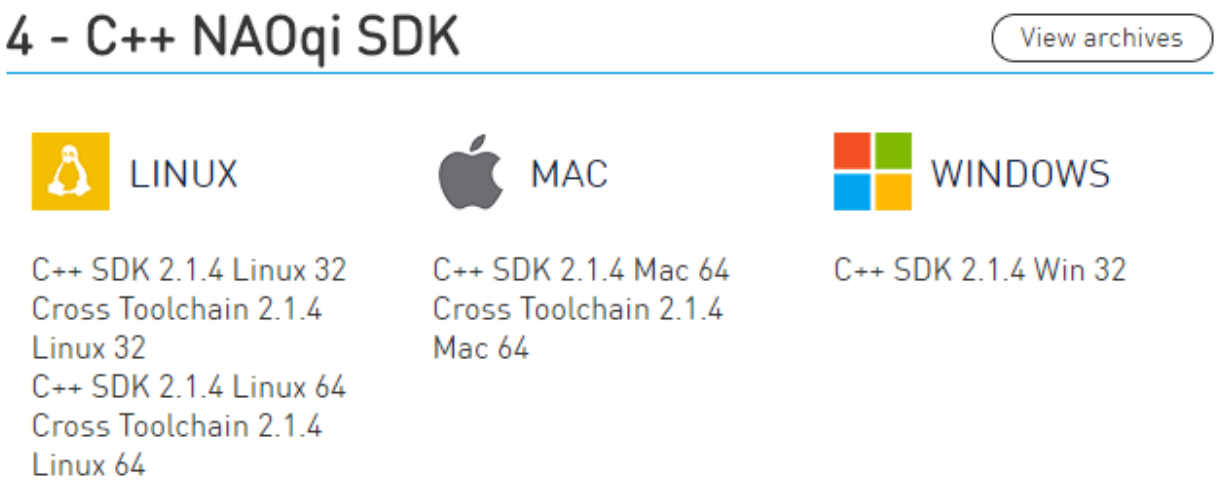


Ilustración 69 Diferentes instaladores de C++ NAOqi SDK.

Una vez descargado nuestro C++ SDK, debemos extraer todo su contenido en un fichero con el nombre naoqi-sdk. Luego creamos una carpeta nueva con el nombre SDKfolder. A continuación, abriremos una terminal, nos dirigiremos a la carpeta SDKfolder y escribiremos:

```
qibuild init
```

Por último, copiaremos la carpeta naoqi-sdk dentro de nuestro SDKfolder.

Como comprobación, abrimos un terminal y ejecutamos el siguiente comando:

```
cd /path/to/SDKfolder/naoqi-sdk/doc/dev/cpp/examples
```

A continuación, creamos la toolchain ejecutando lo siguiente:

```
qitoolchain create mytoolchain /path/to/SDKfolder/naoqi-sdk/toolchain.xml
```

mytoolchain puede ser el nombre que queramos.

Establecemos la configuración para nuestra nueva toolchain y la definimos como default para toda el área de trabajo mediante el siguiente comando:

```
qibuild add-config mytoolchain -t mytoolchain –default
```

Por último, ejecutamos las siguientes líneas para visualizar el proyecto sayhelloworld:

```
cd core/sayhelloworld

qibuild configure

qibuild make
```

7.1.1.3.2 Ubuntu

En cuanto a Ubuntu, deberemos tener instalado la versión 4.4 de GCC o superior o una versión reciente de QtCreator.

También necesitaremos instalar qiBuild en Ubuntu, para ello deberemos cumplir unos requisitos previos. Por un lado, instalar la versión 2.8.3 o superior de CMake, para ello usaremos los paquetes de CMake que nos proporciona nuestra distribución.

Para instalar qiBuild usaremos la herramienta pip. Para instalar la herramienta pip ejecutamos:

```
python get-pip.py.
```

Una recomendación es utilizar la opción --user, para mantener nuestro sistema ordenado. Solo tenemos que escribir en una terminal de Ubuntu lo siguiente:

```
pip install qibuild --user
```

A continuación tenemos que añadir **$HOME/.local/bin** a nuestro **$PATH**. Una forma de añadirlo, es escribir **PATH=$PATH:$HOME/.local/bin** en el ~/.bashrc, al final del archivo y por último guardar las modificaciones realizadas.

Una vez realizados estos pasos, debemos configurar qiBuild. Para ello ejecutar el siguiente comando:

```
qibuild config --wizard
```

En este momento tendremos que especificar el path de CMake, el generador de CMake y el IDE que usaremos. De todas formas podremos modificar la configuración en cualquier momento que queramos solo con volver a ejecutar el comando anterior.

Una vez descargado nuestro C++ SDK, debemos extraer todo su contenido en un fichero con el nombre naoqi-sdk. Luego creamos una carpeta nueva con el nombre SDKfolder. A continuación, abriremos una terminal, nos dirigiremos a la carpeta SDKfolder y escribiremos `$ qibuild init`. Por último, copiaremos la carpeta naoqi-sdk dentro de nuestro SDKfolder.

Como comprobación, abrimos un terminal y ejecutamos el siguiente comando:

```
cd /path/to/SDKfolder/naoqi-sdk/doc/dev/cpp/examples
```

A continuación, creamos la toolchain ejecuantando lo siguiente:

```
qitoolchain create mytoolchain /path/to/SDKfolder/naoqi-sdk/toolchain.xml
```

mytoolchain puede ser el nombre que queramos.

Establecemos la configuración para nuestra nueva toolchain y la definimos como default para toda el área de trabajo mediante el siguiente comando:

```
qibuild add-config mytoolchain -t mytoolchain –default
```

Por último, ejecutamos las siguientes líneas para visualizar el proyecto sayhelloworld:

```
cd core/sayhelloworld

qibuild configure

qibuild make
```

7.1.1.4 Instalación Jetty

Lo primero que debemos ejecutar en un terminal es la siguiente orden:

```
sudo apt-get install openjdk-7-jdk
```

Previamente podemos ejecutar java -version para comprobar que versión de java tenemos instalado en nuestro ordenador.

Ahora debemos introducir la variable de entorno JAVA_HOME, para ello debemos escribir en el terminal lo siguiente:

```
sudo gedit /etc/enviroment
```

Se nos abrirá un documento de texto donde escribiremos JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64.

Una vez hecho esto, guardamos y cerramos el documento. Nos dirigimos de nuevo al terminal y escribimos:

```
source /etc/environment
```

Ahora debemos añadir estas variables de entorno a nuestro bashrc y profile:

```
sudo gedit ~/.bashrc
sudo gedit ~/.profile
```

Escribimos lo siguiente en ambos documentos y lo guardamos.

```
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
export PATH=$JAVA_HOME/bin:$PATH
```

De nuevo en el terminal, introducimos:

```
source ~/.bashrc
source ~/.profile
```

Para comprobar que todos los pasos anteriores se han realizado correctamente, escribimos en el terminal:

```
env | grep JAVA
```

Si como resultado en el terminal nos sale la ruta que le especificamos previamente en el bashrc, todo ha ido bien.

```
JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
```

Para instalar maven solo hay que ejecutar la siguiente línea

```
sudo apt-get install maven
```

Para comprobar la versión de maven que hemos instalado ejecutaremos

```
mvn -version
```

Cuya salida debe ser:

```
Apache Maven 3.0.5
Maven home: /usr/share/maven
Java version: 1.7.0_111, vendor: Oracle Corporation
Java home: /usr/lib/jvm/java-7-openjdk-amd64/jre
Default locale: es_ES, platform encoding: UTF-8
OS name: "linux", version: "4.4.0-45-generic", arch: "amd64", family:
"unix"
```

Importante: Ejecutar una primera vez maven para que descargue paquetes necesarios, por tanto, es imprescindible tenerlo conectado a internet.

7.1.1.5 Instalación ROS

Primero, será necesario instalar la versión de Ubuntu 14,04 LTS en el PC que controlará el Turtlebot. Asegurarse de crear una partición "Swap" de al menos 2GB.

http://www.ubuntu-guia.com/2012/04/como-instalar-ubuntu.html

Una vez instalado, será necesario instalar ROS en su versión "indigo" según se indica en el siguiente enlace, con la diferencia de que el espacio de trabajo se llamará "catkin_ws":

http://wiki.ros.org/indigo/Installation/Ubuntu

Todos los pasos siguientes se realizarán en un terminal de Ubuntu, para abrirlo pulsamos la combinación de teclas Ctr + Alt + T.

7.1.1.5.1 Configurar los repositorios de Ubuntu

Buscamos el "Centro de software de Ubuntu", en la esquina superior izquierda de la pantalla buscamos el menú "Editar" y hacemos clic en "Orígenes del software..."

Al hacer clic sale un menú con diferentes opciones, seleccionamos las que dicen "restricted", "universe" y "multiverse"

7.1.1.5.2 Permisos packages.ros.org

El siguiente paso es permitir que nuestro ordenador acepte software de packages.ros.org, para ello tendremos que ejecutar lo siguiente en la línea de comandos:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu (lsb_release -
sc) main" >    /etc/apt/sources.list.d/ros-latest.list'

sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net --recv-
key 0xB01FA116
```

Una vez realizada la configuración de aceptación de paquetes, es importante asegurarse de que todo esté actualizado y para ello hay que ejecutar lo siguiente:

```
sudo apt-get update
```

7.1.1.5.3 Instalación ROS

Pasamos a la instalación de ROS en Ubuntu. Se recomienda instalar la versión ROS Indigo-Desktop-Full, para ello tendremos que introducir en el terminal la siguiente orden:

```
sudo apt-get install ros-indigo-desktop-full
```

7.1.1.5.4 rosdep

Ya hemos instalado ROS Indigo, pero tenemos que hacer unas configuraciones previas. Lo primero que tenemos que inicializar es "rosdep". rosdep nos permite instalar dependencias del sistema sin problemas y es requerido para algunos componentes de ROS. Para ello tendremos que ejecutar:

```
sudo rosdep init

rosdep update
```

7.1.1.5.5 Variables de entorno de ROS

El siguiente paso es añadir las variables de entorno de ROS a nuestro bashrc y profile para que se ejecuten cada vez que iniciemos un terminal o se llame desde la aplicación respectivamente.
Para ello, ejecutar los siguientes comandos:

```
echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc

source ~/.bashrc

echo "source /opt/ros/indigo/setup.bash" >> ~/.profile

source ~/.profile
```

Nota: Es importante saber que aquellos terminales que se encontraban abiertos en Ubuntu en el momento de hacer source de bashrc en otro terminal no van a cargar correctamente el .bashrc, pero los nuevos terminales que se abran después de esto sí cargan.

7.1.1.5.6 rosinstall

Uno de los pasos finales en el proceso de instalación de ROS Indigo en Ubuntu 14.04 es instalar “rosinstall” que no es más que una herramienta de la línea de comandos que nos permite descargar paquetes de ROS con un solo comando

```
sudo apt-get install python-rosinstall
```

7.1.1.5.7 Creando un ROS workspace

A continuación, creamos el workspace de ROS que usaremos para instalar nuestros paquetes:

```
mkdir -p catkin_ws/src

cd ~/catkin_ws/src

catkin_init_workspace
```

7.1.1.5.8 Añadiendo el workspace al bashrc

Una vez creado, compilamos el workspace:

```
cd ~/catkin_ws

catkin_make
```

Obteniendo algo como esto por el terminal:

Base path: /home/username/catkin_ws
Source space: /home/username/catkin_ws/src
Build space: /home/username/catkin_ws/build
Devel space: /home/username/catkin_ws/devel
Install space: /home/username/catkin_ws/install
####
Running command: "cmake /home/username/catkin_ws /src -DCATKIN_DEVEL_PREFIX=/home/username/catkin_ws /devel -DCMAKE_INSTALL_PREFIX=/home/username/catkin_ws /install -G Unix Makefiles" in "/home/username/catkin_ws /build"
####
-- The C compiler identification is GNU 4.8.4
-- The CXX compiler identification is GNU 4.8.4
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done

```
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Using CATKIN_DEVEL_PREFIX: /home/username/catkin_ws/devel
-- Using CMAKE_PREFIX_PATH: /home/username/catkin_ws/devel;/opt/ros/indigo
-- This workspace overlays: /home/username/catkin_ws/devel;/opt/ros/indigo
-- Found PythonInterp: /usr/bin/python (found version "2.7.6")
-- Using PYTHON_EXECUTABLE: /usr/bin/python
-- Using Debian Python package layout
-- Using empy: /usr/bin/empy
-- Using CATKIN_ENABLE_TESTING: ON
-- Call enable_testing()
-- Using CATKIN_TEST_RESULTS_DIR: /home/username/catkin_ws/build/test_results
-- Looking for include file pthread.h
-- Looking for include file pthread.h - found
-- Looking for pthread_create
-- Looking for pthread_create - not found
-- Looking for pthread_create in pthreads
-- Looking for pthread_create in pthreads - not found
-- Looking for pthread_create in pthread
-- Looking for pthread_create in pthread - found
-- Found Threads: TRUE
-- Found gtest sources under '/usr/src/gtest': gtests will be built
-- Using Python nosetests: /usr/bin/nosetests-2.7
-- catkin 0.6.18
-- BUILD_SHARED_LIBS is on
-- Configuring done
-- Generating done
-- Build files have been written to: /home/username/catkin_ws/build
####
#### Running command: "make -j4 -l4" in "/home/username/catkin_ws/build"
####
```

Y tras ello, modificamos tanto el bashrc como el profile para indicarles el directorio del workspace creado.

```
gedit ~/.bashrc
```

Vamos al final del archivo y añadimos:

```
source ~/catkin_ws/devel/setup.bash
```

Realizamos **source ~/.bashrc** en un terminal y comprobamos que se ha incluido correctamente ejecutando **echo $ROS_PACKAGE_PATH**, obteniendo lo siguiente:

```
/home/”USER_NAME”/catkin_ws/src:/opt/ros/indigo/share:/opt/ros/indigo/
stacks
```

Realizaremos la misma operación con el profile:

```
gedit ~/.profile
```

Vamos al final del archivo y añadimos:

```
source ~/catkin_ws/devel/setup.bash
```

7.1.2 Funcionamiento

Para ejecutar todo el código debemos realizar lo siguiente. Abrimos 3 terminales de Linux, ya sea usando Ctrl+Alt+T o pulsando 3 veces en el icono de la izquierda. Tenemos que asegurarnos de que nuestro ordenador y el robot se encuentran en la misma red. Después de esta comprobación, abrimos el navegador Mozilla Firefox y ejecutamos el plugin HttpRequester.

Una vez realizado todo lo anterior, nos dirigimos a uno de los 3 terminales abiertos y escribimos lo siguiente:

```
cd "carpeta_donde_este_guardado_el_servidor"/MinimalServelts

mvn exec:java
```

Como ejemplo, según la estructuras de ficheros que yo tengo, la ejecución del servidor se realizaría de la siguiente manera:

```
cd ~/Escritorio/Nao/Maven/Server1/MinimalServelts

mvn exec:java
```

El segundo servidor se ejecutaría de la misma manera, solo tendríamos que cambiar Server1 por Server 2:

```
cd ~/Escritorio/Nao/Maven/Server1/MinimalServelts

mvn exec:java
```

Nota: Si se van a utilizar los 2 servidores que hice, recomiendo pegar la carpeta Maven, que facilito, en la carpeta personal de Ubuntu, dicha carpeta Maven contendrá a los 2 servidores, de esta forma la ejecución de los servidores será de la siguiente forma:

```
cd ~/Maven/Server1/MinimalServlets

mvn exec:java
```

Para el servidor 2 se haría igual, pero habría que sustituir Server2 por Server1 en la orden cd.

Y el código del robot se ejecuta de la siguiente manera:

```
cd ~/pynaoqi-python2.7-2.1.4.13-linux64

python Mercurio.py
```

Una vez ejecutado todo eso en el terminal, tendremos los 2 servidores operativos y el robot esperando a hablar o a recibir un código QR. Los servidores serán similares a la siguiente imagen si todo salió bien.

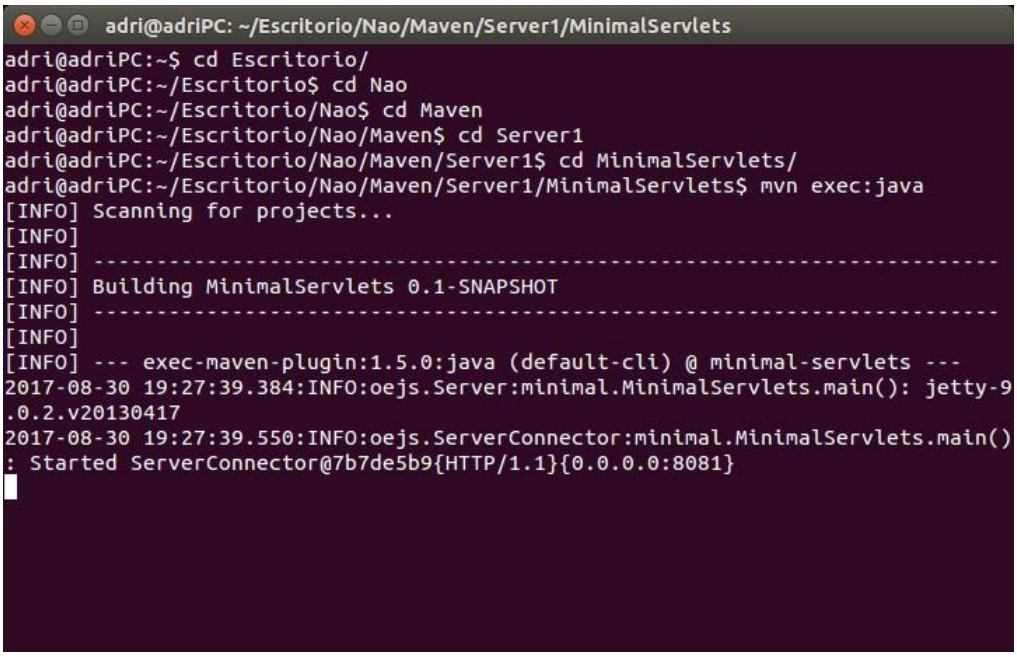


Ilustración 70 Servidor operativo.

Los dos servidores serán simulares, la única diferencia será el último número, en la imagen 8081, que es al puerto al que está enlazado. Por todo lo demás, son idénticos. El terminal en el que se ejecutará el Nao no mostrará algo demasiado diferente a la ejecución de cualquier programa de Python den Linux, solo se podrá cierta información, tal y como ID del cliente o frases que va a decir el robot. Por último, tendremos la ventana de HttpRequester:

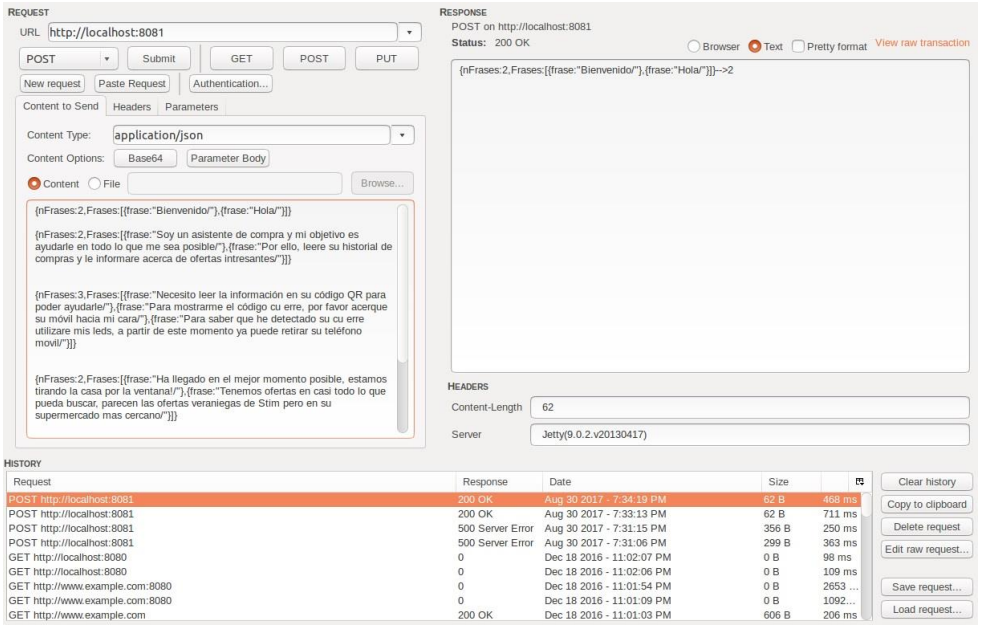


Ilustración 71 HttpRequester.

Para enviarle un texto para que lo diga el Nao, se usará el HTTPRequester siguiendo la estructura de mensaje:

{**nFrases**: ,**Frases**:[{**frase**:"/"}]}

Ejemplo:

{**nFrases**: 2,**Frases**:[{**frase**:"Bienvenido/",**frase**:"Soy un asistente de compra y mi objetivo es ayudarle en todo lo que me sea posible/"}]}

En este ejemplo nFrases son el número total de frases, Frases, es el conjunto de frases y frase es la cabecera de cada una.

Se eligió esta estructura ya que el Turtlebot también estaba en este proyecto y su comunicación estaba configurado con la misma configuración.

Posteriormente el código de Python se encargará de obtener las frases, si utilizamos las del ejemplo tendríamos lo siguiente:

Frase1: "Bienvenido".

Frase2=" Soy un asistente de compra y mi objetivo es ayudarle en todo lo que me sea posible".

El mensaje en forma JSON se escribirá en el panel de la izquierda, tal y como se ve en la ilustración 58, a continuación en el campo URL se escribe la dirección del servidor en este caso http://localhost:8081. Haciendo clic en POST el mismo HttpRequester nos informará si el mensaje llegó correctamente, aunque en los diferentes terminales, en el del servidor del robot y en el del robot, se podrán visualizar diferentes prints con dicha información.

Para leer el QR, cuya estructura será el nombre del cliente entre "", lo único que hay que hacer es ponerle el código delante (se puede ver si está leyendo lo que pone en el QR porque lo va mostrando por el terminal de Linux). Luego, el código que ejecuta el robot se encarga de enviar la ID obtenida al servidor del robot (Server1, puerto 8081) y este de estructurarla en forma de JSON y enviarla al servidor de la empresa (Server2, puerto 9000). La estructura será bastante sencilla:

{**identificador**: "ID del cliente"}

Tanto la lectura de QR como que reproduzca el texto que se le envíe se ejecutará según se le envíen o el robot detecte el código. Por ejemplo, si el robot está detectando QR se le puede enviar un texto para que lo diga. Ninguna de las dos acciones está ligada en ejecución. Se puede interpretar el texto indistintamente del QR y viceversa.

Por lo tanto, el esquema de comunicaciones del robot será el siguiente:

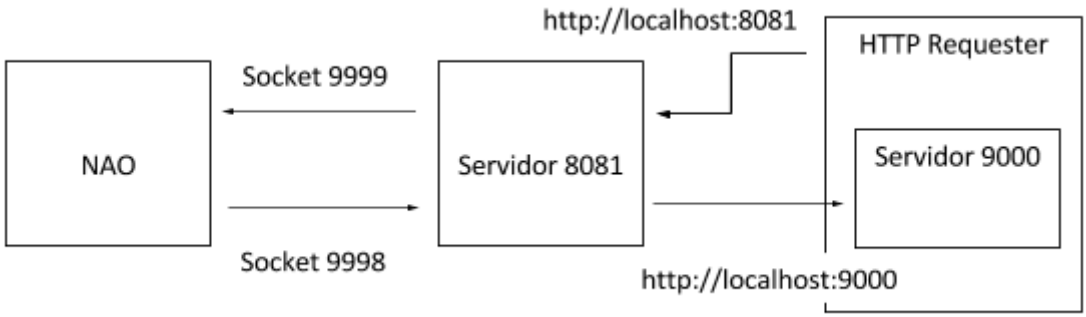


Ilustración 72 Esquema de comunicación.

7.2 Programas

7.2.1 Conocer version NAO

```
#!/usr/bin/env python

import argparse
from naoqi import ALProxy

def main(robotIP, PORT=9559):
    motionProxy = ALProxy("ALMotion", robotIP, PORT)

    # Example showing how to get the robot config
    robotConfig = motionProxy.getRobotConfig()
    for i in range(len(robotConfig[0])):
        print robotConfig[0][i], ": ", robotConfig[1][i]

    # "Model Type"   : "naoH25", "naoH21", "naoT14" or "naoT2".
    # "Head Version" : "VERSION_32" or "VERSION_33" or "VERSION_40".
    # "Body Version" : "VERSION_32" or "VERSION_33" or "VERSION_40".
    # "Laser"        : True or False.
    # "Legs"         : True or False.
    # "Arms"         : True or False.
    # "Extended Arms": True or False.
    # "Hands"        : True or False.
    # "Arm Version"  : "VERSION_32" or "VERSION_33" or "VERSION_40".
    # Number of Legs : 0 or 2
    # Number of Arms : 0 or 2
    # Number of Hands: 0 or 2


if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--ip", type=str, default="127.0.0.1",
```

```
                        help="Robot ip address")
    parser.add_argument("--port", type=int, default=9559,
                        help="Robot port number")

    args = parser.parse_args()
    main(args.ip, args.port)
```

7.2.2 Código Nao

El programa a continuación es el que he desarrollado para cumplir con las especificaciones de la empresa. Este código es el resultado de todo lo que aprendí. Como puede verse a lo largo del código, he utilizado diferentes funciones para aunar distintas acciones como también me he servido de la programación orientada a objetos. Otra cosa a destacar del robot es que trabajar por eventos, es decir, intenta emular nuestras reacciones cuando se nos provee información, de tal forma, el evento ALBarcodeReader estará siempre a la espera de poder leer un QR y cuando lo haga le enviará la información al servidor que emula la empresa pasando previamente por su propio servidor y estructurando el mensaje según el formato JSON. Esto no impide que en cualquier momento seamos capaces de enviarle unas frases para que las diga al cliente. Ya que la acción de leer el QR se encuentra en un segundo plano.

```
#!/usr/bin/env python

from naoqi import *
import time
import socket

ip = "169.254.87.118"
port = 9559

broker = ALBroker("pythonBroker","0.0.0.0", 0, ip, port)
motion = ALProxy("ALMotion", ip, port)
memory = ALProxy("ALMemory", ip, port)
basicawareness = ALProxy("ALBasicAwareness", ip, port)
barcode = ALProxy("ALBarcodeReader", ip, port)
tts = ALProxy("ALTextToSpeech", ip, port)
aas = ALProxy("ALAnimatedSpeech", ip, port)
memory = ALProxy("ALMemory", ip, port)
leds = ALProxy("ALLeds", ip, port)

def compruebaCabeza(ip, port):
    motion = ALProxy("ALMotion", ip, port)
    names = "Head"
    useSensors = False
    commandAngles = motion.getAngles(names, useSensors)
    print "Command angles: "
    print str(commandAngles)
    print ""
    return commandAngles

def mueveCabeza(ip, port, objetive):
    motion = ALProxy("ALMotion", ip, port)
    motion.setStiffnesses("Head", 1.0)
```

```
    names = "HeadYaw"
        angles = objetive[0]
    fractionMaxSpeed = 0.1
    motion.setAngles(names,angles,fractionMaxSpeed)
        names = "HeadPitch"
        angles = objetive[1]
        fractionMaxSpeed = 0.1
        motion.setAngles(names,angles,fractionMaxSpeed)

def manejaEstimulos(ip, port, cad):
    basicawareness = ALProxy("ALBasicAwareness", ip, port)
    basicawareness.setStimulusDetectionEnabled("People", cad)
    basicawareness.setStimulusDetectionEnabled("Movement", cad)
    basicawareness.setStimulusDetectionEnabled("Touch", cad)
    basicawareness.setStimulusDetectionEnabled("Sound", cad)

class misocket:

    def __init__(self):
    self.sock = socket.socket()

    def conecta(self, host, port):
        self.sock.connect((host, port))

    def escucha(self, host, port):
    self.sock.bind((host, port))
    self.sock.listen(10)
    print "Ahora escucha"

    def envia(self,msg):
    msg ="ID: " + msg + " FIN"
    self.sock.sendall(msg)

    def recibe(self):
    conn, addr = self.sock.accept()
    print ("Conexion establecido con ", addr)
    self.msg = conn.recv(1024)
    return True

    def dameMensaje(self):
    return self.msg

    def cierra(self):
    print "Cerrando conexion..."
    self.sock.close()

class myEventHandler(ALModule):

    def myCallback(self, key, value, msg):

    print (key, value, msg)

    if(value):
```

```
            memory.post.insertData("cadena", value)
            cad = str(value)
            aux = cad.split("'",3)
            aux = aux[1]
            print aux

            listavacia.append(aux)
            print (listavacia)
            n = len(listavacia)
            print n

            OldID = listavacia[n-2]
            print OldID
            CurrentID = listavacia[n-1]
            print CurrentID

            if(CurrentID != OldID):
                print "Es diferente"
                msg = "Bienvenido" + CurrentID
                habla(msg, ip, port)
                NaoSocket = misocket()
                NaoSocket.conecta("localhost", 9998)
                NaoSocket.envia(aux)
                NaoSocket.cierra()

            else:
                print "No es diferente"


        else:
            memory.post.insertData("nombre", "")

    def habla(msg, ip, port):
        tts = ALProxy("ALTextToSpeech", ip, port)
        id = tts.post.say(msg)
        return id

    if __name__ == "__main__":
        manejaEstimulos(ip, port, False)
        #objetivo = [0.0, 0.51]
        objetivo = [-0.0031099319458007812, 0.09199810028076172]
        mueveCabeza(ip, port, objetivo)
        time.sleep(2)
        listavacia = []
        listavacia.append("Nada")
        n = 0
        NaoSocketServer = misocket()
        NaoSocket = misocket()
        NaoSocketServer.escucha("localhost", 9999)
        calculado = compruebaCabeza(ip, port)
        error = [objetivo[0] - calculado[0], objetivo[1] - calculado[1]]

        print 'Errores: '
```

```
    print str(error)
    print ''

    handlerModule = myEventHandler("handlerModule")
    while True:
        id =
memory.post.subscribeToEvent("BarcodeReader/BarcodeDetected",
"handlerModule", "myCallback")
        bol = NaoSocketServer.recibe()
        if(bol == True):
            memory.stop(id)
            frases = NaoSocketServer.dameMensaje()
            frases = frases[2:len(frases)]

            frases = frases.split("/")
            print frases
            iteracion = 0
            for iteracion in range(0,len(frases)):
                #aas.say(str(frases[iteracion]))
                tts.say(str(frases[iteracion]))
                iteracion = iteracion + 1

            memory.post.insertData("cadena", '')
```

7.2.3 Programa exploración Turtlebot

El inicio con ROS fue bastante duro, ya que tienes que tener en mente demasiadas variables. Fue gracias a los compañeros en el proyecto que tuve un menor tiempo de aprendizaje. Ellos ya habían pasado mucho tiempo progamando de esta forma, tanto fue así que enfocaron mi aprendizaje en este campo. De esta forma, tardé mucho menos tiempo. Una vez que ya entendía como trabajar en ROS me delegaron la tarea de mejorar la tarea de exploración del Turtlebot. Tras varios intentos y después de algunas correcciones por parte de los demás miembros del proyecto, este el resultado. El código anterior de exploración funcionaba pero algunas veces cuando el Turtlebot se le pedía que se parase por que el operario lo demandaba mediante la interfaz, el robot no obedecía. Este código resolvió todos los problemas.

```
#include "iostream"
#include <ros/ros.h>
#include <std_srvs/Trigger.h>
#include <cstdlib>
#include <ros/ros.h>
#include <actionlib/client/simple_action_client.h>
#include <nav2d_navigator/ExploreAction.h>
#include <nav2d_navigator/commands.h>

using namespace std;

#define TIMEOUT 240
```

```
typedef actionlib::SimpleActionClient<nav2d_navigator::ExploreAction>
ExploreClient;

ExploreClient* gExploreClient;

class Services
{
public:
        Services();
        std_srvs::Trigger srv;
        bool metCancela(std_srvs::Trigger::Request &req,
std_srvs::Trigger::Response &res);
        bool metInicio(std_srvs::Trigger::Request &req,
std_srvs::Trigger::Response &res);
        bool receiveCommand(std_srvs::Trigger::Request &req,
std_srvs::Trigger::Response &res);
        void MaquinaEstados();
        void Exito();

private:
        ros::NodeHandle n,c,v,ex;
        ros::ServiceServer cancel, start;
        ros::ServiceClient victory;
        ros::ServiceServer cmdServer;
        int estado;// connected;
        bool aborted;
        double inicio, total;
};

Services::Services():
estado(0)
{
        start  = n.advertiseService("metodo_Inicio",
&Services::metInicio, this);
        cancel = c.advertiseService("metodo_Cancela",
&Services::metCancela, this);
        victory= v.serviceClient<std_srvs::Trigger>("metodo_Exito");
        cmdServer = ex.advertiseService(NAV_EXPLORE_SERVICE,
&Services::receiveCommand,this);

        gExploreClient = new ExploreClient(NAV_EXPLORE_ACTION, true);

}

bool Services::receiveCommand(std_srvs::Trigger::Request &req,
std_srvs::Trigger::Response &res)
{
        nav2d_navigator::ExploreGoal goal;
        gExploreClient->sendGoal(goal);
        inicio = ros::Time::now().toSec();
        while (true)
        {
                total = ros::Time::now().toSec() - inicio;
```

```
                if(gExploreClient->getState() ==
actionlib::SimpleClientGoalState::SUCCEEDED)
                {
                        Exito();
                        ROS_INFO("Objetivo conseguido");
                        estado = 1;
                        res.success = true;
                        return true;
                }

                if(aborted == true)
                {
                        ROS_WARN("Exploracion abortada");
                        estado = 2;
                        aborted = false;
                        return true;

                }

                if(total >= TIMEOUT)
                {

                        gExploreClient->cancelAllGoals();
                        ROS_ERROR("Demasiado tiempo realizando la
exploracion");
                        return false;
                }
        ros::spinOnce();
        }
}

bool Services::metInicio(std_srvs::Trigger::Request &req,
std_srvs::Trigger::Response &res)
{
        res.success = true;
        res.message = "Conexion establecida";
        return true;
}

bool Services::metCancela(std_srvs::Trigger::Request &req,
std_srvs::Trigger::Response &res)
{
        gExploreClient->cancelAllGoals();
        aborted = true;
        res.success = true;
        res.message = "Cancelada";
        return true;
}

void Services::Exito()
{
        if(victory.call(srv))
```

```
        {

        }
}

void Services::MaquinaEstados()
{
        while(ros::ok())
        {
                switch(estado)
                {
                        case 0:
                                gExploreClient->waitForServer();
                                break;

                        case 1:
                                //ROS_INFO("Exploracion terminada");
                                estado = 0;
                                break;

                        case 2:
                                //ROS_INFO("Cancelada");
                                estado = 0;
                                break;

                }
        ros::spinOnce();
        }
}


int main(int argc, char **argv)
{
        ros::init(argc, argv, "Nodo_Explora");
        ROS_INFO("Se inicio el nodo explora");
        Services servicios;
        servicios.MaquinaEstados();

        delete gExploreClient;

}
```

7.2.4 Servidor NAO

```
package minimal;

import java.io.IOException;
import java.io.PrintWriter;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.*;
import java.net.*;
import java.net.HttpURLConnection;
import java.net.URL;
import org.eclipse.jetty.util.Fields;
import org.eclipse.jetty.util.Fields.Field;
import java.util.List;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.net.ssl.HttpsURLConnection;

import org.eclipse.jetty.server.Server;
import org.eclipse.jetty.servlet.ServletHandler;
import org.eclipse.jetty.client.HttpClient;
import org.eclipse.jetty.client.HttpExchange;
import org.eclipse.jetty.client.api.ContentResponse;

import org.json.JSONArray;
import org.json.JSONObject;
import org.json.JSONException;
import org.json.JSONTokener;
import org.json.HTTP;


import java.text.ParseException;


public class MinimalServlets

{       static boolean info_nao_enviar;
        public static void sendPost(String cadena) throws Exception
        {
        try
        {
                //PrintWriter out = resp.getWriter();
                JSONObject object = new JSONObject();
                object.put("identificador", cadena);
                //System.out.println(object);
        // Send data
                URL url = new URL("http://localhost:9000");
                URLConnection conn = url.openConnection();
                conn.setDoOutput(true);
```

```
                OutputStreamWriter wr = new
OutputStreamWriter(conn.getOutputStream());
                wr.write(object.toString());
                wr.flush();

        // Get the response
                BufferedReader rd = new BufferedReader(new
InputStreamReader(conn.getInputStream()));
                System.out.println(rd);
                String line;
                        while ((line = rd.readLine()) != null) {
        // Process line...

                        }
                System.out.println(rd);
                wr.close();
                rd.close();

                System.out.println(info_nao_enviar);
                info_nao_enviar = false;
                System.out.println(object);
                //out.print(object);
        }
        catch (Exception e) {
                e.printStackTrace();}


        }


        public static void enviaMensaje(String cadena)
        {
        try{
                Socket soc= new Socket("localhost", 9999);
                DataOutputStream dout= new
DataOutputStream(soc.getOutputStream());
                dout.writeUTF(cadena);
                System.out.println(cadena);
                System.out.println(info_nao_enviar);
                info_nao_enviar = false;
                System.out.println(info_nao_enviar);
                dout.flush();
                dout.close();
                soc.close();
        }
        catch(Exception e){
                 e.printStackTrace();}
        }

        public static void escuchaYenvia()
        {
        try
        {
```

```
                ServerSocket soc2 = new ServerSocket(9998);
                Socket MiServicio = soc2.accept();
                System.out.println("Estableciendo conexion...");
                System.out.println(MiServicio);
                BufferedReader entrada = new BufferedReader(new
InputStreamReader(MiServicio.getInputStream()));
                String mensajeRecibido = entrada.readLine();
                entrada.close();
                int longitud = mensajeRecibido.length();
                //System.out.println(longitud);
                String id = mensajeRecibido.substring(0,3);
                String recorte = mensajeRecibido.substring(longitud-
3,longitud);
                String informacion =
mensajeRecibido.substring(id.length(), longitud - recorte.length() -
1);
                //String norecorte =
mensajeRecibido.substring(0,longitud-3);
                System.out.println(info_nao_enviar);
                System.out.println(id);
                System.out.println(informacion);
                System.out.println(recorte);
                if(recorte.equals("FIN"))
                {

                soc2.close();
                info_nao_enviar = true;
                MinimalServlets.sendPost(informacion);
                }

                //soc2.close();
        }
        catch(Exception e){
                 e.printStackTrace();}
        }


    public static void main( String[] args ) throws Exception
    {


        //El servidor escucha en el puerto 8081
        Server server = new Server(8081);


        ServletHandler handler = new ServletHandler();
        server.setHandler(handler);

       handler.addServletWithMapping(HelloServlet.class, "/*");
       info_nao_enviar = false;
        server.start();
       //server.join();
       while(true)
```

```
        {
        MinimalServlets.escuchaYenvia();
        }

        //server.join();

    }


    @SuppressWarnings("serial")
    public static class HelloServlet extends HttpServlet
    {


        @Override
        protected void doGet( HttpServletRequest request,
                              HttpServletResponse response ) throws
ServletException,

IOException
        {
            response.setContentType("text/html");
            response.setStatus(HttpServletResponse.SC_OK);
            response.getWriter().println("<h1>Hello from
HelloServlet</h1>");

        }

        @Override
        protected void doPost(HttpServletRequest req,
HttpServletResponse resp)
            throws ServletException, IOException
        {


        PrintWriter out = resp.getWriter();
        if(info_nao_enviar == false)
        {
        //SACAR CAMPOS DE JSON
                BufferedReader br = new BufferedReader(new
InputStreamReader(req.getInputStream()));
                String json = "";
                if(br != null){
                        json = br.readLine();
                out.print(json);}
                String DatosAPython = "";
                JSONObject jsonObject;
                jsonObject = new JSONObject(json);
                int nfrases = jsonObject.getInt("nFrases");
                JSONArray jarray;
                jarray = jsonObject.getJSONArray("Frases");
                JSONObject nuevasf;
                String[] aux = new String[nfrases];
```

```
                try{
        //JSONObject jsonObject = org.json.HTTP.toJSONObject(json);
        //jsonObject = new JSONObject(json);
                out.print("-->"+jsonObject.get("nFrases"));

                } catch (JSONException e) {
                throw new IOException("Prueba 1 Error parsing JSON
request string");
        }
        //MinimalServlets.enviaMensaje(jsonObject.getString("name"));
        // {nFrases:2,Frases:[{frase:"Bienvenido/"},{frase:"Hola/"}]}

                for (int n=0;n<nfrases;n++)
                {
                        nuevasf = (JSONObject)jarray.get(n);
                        aux[n] = nuevasf.getString("frase");
                        DatosAPython = DatosAPython + aux[n];
                }

        MinimalServlets.enviaMensaje(DatosAPython);
        info_nao_enviar = false;

        //MinimalServlets.escuchaYenvia();

        }

        if (info_nao_enviar == true)
        {
                //JSONObject object = new JSONObject();
                System.out.println("Entro en el POST");
        }


         }

    }

}
```

7.2.5 Servidor simulado empresa

```
package minimal;

import java.io.IOException;
import java.io.PrintWriter;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.*;
import java.net.*;

import java.util.List;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.eclipse.jetty.server.Server;
import org.eclipse.jetty.servlet.ServletHandler;

import org.json.JSONArray;
import org.json.JSONObject;
import org.json.JSONException;
import org.json.JSONTokener;
import org.json.HTTP;


import java.text.ParseException;


public class MinimalServlets

{

    public static void main( String[] args ) throws Exception
    {



        Server server = new Server(9000);


        ServletHandler handler = new ServletHandler();
        server.setHandler(handler);

       handler.addServletWithMapping(HelloServlet.class, "/*");

        server.start();
       server.join();
```

```
    }


    @SuppressWarnings("serial")
    public static class HelloServlet extends HttpServlet
    {


        @Override
        protected void doGet( HttpServletRequest request,
                              HttpServletResponse response ) throws
ServletException,

IOException
        {
            response.setContentType("text/html");
            response.setStatus(HttpServletResponse.SC_OK);
            response.getWriter().println("<h1>Hello from
HelloServlet</h1>");

        }

        @Override
        protected void doPost(HttpServletRequest req,
HttpServletResponse resp)
            throws ServletException, IOException
        {
        PrintWriter out = resp.getWriter();
        BufferedReader br = new BufferedReader(new
InputStreamReader(req.getInputStream()));
                String json = "";
                if(br != null){
                        json = br.readLine();
                //out.print(json);
                }
                System.out.println(json);
                JSONObject jsonObject;
                jsonObject = new JSONObject(json);
                String id = jsonObject.getString("identificador");

        System.out.println(req);
        System.out.println(resp);
        System.out.println(id);
        System.out.println("\n");
        //BufferedReader wr = new BufferedReader(new
InputStreamReader(conn.getOutputStream()));
        /*String clientIP = req.getRemoteAddr();
        String clientHOST = req.getRemoteHost();
         resp.setContentType("text/plain");
         PrintWriter out = resp.getWriter();
         out.println("IP  : " + clientIP);
         out.println("Host: " + clientHost);*/
```

```
        }


    }
}
```

7.2.6 Navegación estanterias Turtlebot

Una vez que complete la tarea de solucionar el problema de exploración del Turtlebot, junto a otro compañero del proyecto nos dedicamos a desarrollar el software para detectar las estanterías automáticamente a partir de una imagen y luego mandar al robot los objetivos necesarios para que el robot quedase mirando los productos y los fuese reconociendo a la largo de toda la estantería. Tuvimos que hacer una configuración previa que se encuentra dentro del archivo nav_est.h que es el código que se encuentra a continuación de este.

```
#include "nav_est.h"

ObjetivosEstanterias::ObjetivosEstanterias():
                idFlechas(0), idLineas(1), linea(0),
userHome(getenv("HOME")), quitar(false)
{
        //goalPublisher =
gp.advertise<geometry_msgs::PoseStamped>("goal", 1);
        //marker_pub =
n.advertise<visualization_msgs::Marker>("visualization_marker", 1000);
        pubMarkerArray =
nhArray.advertise<visualization_msgs::MarkerArray>("visualization_mark
er_array", 100);
        pubMArrayLines =
nhLines.advertise<visualization_msgs::MarkerArray>("visualization_mark
er_array", 100);
        //servicioObjetivos =
so.serviceClient<etapa_guiado::Objetivo>("consigue_objetivo");

        server = s.advertiseService("objetivos_estanterias",
&ObjetivosEstanterias::mandarObjetivos, this);

        //MOD 24/03/17 Para distancia óptima
        makePlan =
mp.serviceClient<nav_msgs::GetPlan>("/move_base/make_plan");
//Servicio para calcular la distancia a los productos

        initMarkers();

        strcpy(yamlPath, userHome);
        strcat(yamlPath,
"/catkin_ws/src/interfaz/programa_central/files/map.yaml");

        strcpy(pgmPath, userHome);
        strcat(pgmPath,
"/catkin_ws/src/interfaz/programa_central/files/map.pgm");

        src = imread(pgmPath, 0);
        sizeMapa = src.size();
```

```
        //Necesitamos extraer del archivo de metadatos del mapa
(map.yaml) la siguiente información:
        //1. El punto de origen (en metros).
        //2. La resolución del mapa (en metros por píxel).
        origen = obtenerOrigen();
        resolucion = resolucionMapa();
        //Obtenemos los puntos de inicio y fin de cada línea
encontrada.
        puntosEstanterias = estanterias();

        //PROBANDO EN EL CONSTRUCTOR -> NO FUNCIONA
        //enviaRVIZ();
}

void ObjetivosEstanterias::initMarkers()
{
        flechas.header.frame_id="map";
        flechas.header.stamp = ros::Time();
        flechas.ns = "points_and_lines";
        flechas.type = visualization_msgs::Marker::ARROW;
        flechas.action = visualization_msgs::Marker::ADD;
        flechas.scale.x = DIST_EST;
        flechas.scale.y = 0.1;
        flechas.scale.z = 0.1;
        flechas.color.a = 1.0; //No hay que olvidarse del alfa.
        flechas.color.r = 0.0;
        flechas.color.g = 1.0;
        flechas.color.b = 0.0;

        line_list.header.frame_id = "map";
        line_list.header.stamp = ros::Time();
        line_list.ns = "points_and_lines";
        line_list.type = visualization_msgs::Marker::LINE_LIST;
        line_list.action = visualization_msgs::Marker::ADD;
        //LINE_LIST markers use only the x component of scale, for the
line width
        line_list.scale.x = 0.1;
        // Line list is red
        line_list.color.a = 1.0;       //No hay que olvidarse del alfa.
        line_list.color.r = 1.0;
        line_list.color.g = 0.0;
        line_list.color.b = 0.0;
}

//Función que es llamada cuando se pulsa el botón "Enviar" del panel
de Localización de Productos.
bool
ObjetivosEstanterias::mandarObjetivos(localizacion_estanterias::Objeti
vosEstanterias::Request  &req,

          localizacion_estanterias::ObjetivosEstanterias::Response
&res)
{

        //Se obtiene los puntos de aproximación a las líneas y son
preparados para ser mandados. NO FUNCIONA EN EL CONSTRUCTOR.
        enviaRVIZ();

        res.objetivosx.assign (objx.begin(),objx.end());
        res.objetivosy.assign (objy.begin(),objy.end());
```

```
        res.objetivosyaw.assign (objyaw.begin(),objyaw.end());

        objx.clear();
        objy.clear();
        objyaw.clear();

        return true;
}

float ObjetivosEstanterias::resolucionMapa()
{
        string cadena = "";
        string strTemp;
        linea = 0;
        float res = 0.0;

        fstream filein(yamlPath);

        if (!filein)
                cout << "Error abriendo el archivo" << endl;

        //Mientras haya líneas del documento por revisar
        while (getline(filein, strTemp))
        {
                //La resolución del mapa se escribe dentro del yaml
siempre en la línea 2 (1 si contamos la 1ª línea como 0)
                if (linea == 1)
                {
                        //Obtenemos lo que nos interesa de la línea (la
resolución del mapa) y salimos del bucle
                        cadena = strTemp.substr(strTemp.find("
")+1,strTemp.size());
                        break;
                }
                linea++;
        }

        stringstream(cadena) >> res;
        return res;
}

//Obtener las coordenadas del origen del mapa (usado en
obtenerOrigen()), incluyendo comas y espacios en blanco.
string ObjetivosEstanterias::obtenerDatos(string strTemp)
{
        unsigned first = strTemp.find("[");
        unsigned last = strTemp.find("]");
        //Hace falta sumar y restar para que no se incluyan los
corchetes en el substring
        //cout << strTemp.substr (first+1,last-first-1) << endl;
        return strTemp.substr (first+1,last-first-1);
}

vector<float> ObjetivosEstanterias::obtenerOrigen()
{
        string cadena = "";
        string strTemp;
        linea = 0;

        fstream filein(yamlPath);
```

```
        if (!filein)
                cout << "Error abriendo el archivo" << endl;

        //Mientras haya líneas del documento por revisar
        while (getline(filein, strTemp))
        {
                //El punto de origen en el mapa se escribe dentro del
yaml siempre en la línea 3 (2 si contamos la 1ª línea como 0)
                if (linea == 2)
                {
                        //Obtenemos lo que nos interesa de la línea (el
origen) y salimos del bucle
                        cadena = obtenerDatos(strTemp);
                        break;
                }
                linea++;
        }

        return creaOrigen(cadena);
}

//Función que se encarga de obtener las coordenadas del origen,
eliminando espacios en blanco y comas.
vector<float> ObjetivosEstanterias::creaOrigen(string cadena)
{
        vector<float> res;
        vector<string> cadenaSeparada;

        //Separamos los elementos por coma y un espacio, pero sigue
guardando en el vector los espacios.
        boost::split(cadenaSeparada, cadena, boost::is_any_of(", "));
        //El tamaño de "cadenaSeparada" es 5, los elementos impares son
espacios que no nos valen
        //cout << cadenaSeparada.size() << endl;

        for (string componente: cadenaSeparada)
        {
                //Nos quedamos con los elementos en posiciones pares
                if((find(cadenaSeparada.begin(), cadenaSeparada.end(),
componente) - cadenaSeparada.begin()) % 2 == 0)
                {
                        float compFloat = 0.0;
                        //Convertimos el string "componente" en un
float a través de "compFloat" y se añade el float al resultado.
                        stringstream(componente) >> compFloat;
                        res.push_back(compFloat);
                }
        }

        return res;
}

//Pares de puntos que representan el principio y el fin de las
estanterías
vector<vector<geometry_msgs::Point> >
ObjetivosEstanterias::estanterias()
{
        vector<vector<geometry msgs::Point> > paresPuntos;

        if(src.empty())
                cout << "can not open " << pgmPath << endl;
```

```
        Mat dst, cdst;
        //Aplicamos el detector de bordes.
        Canny(src, dst, 50, 200, 3);
        //Invertimos el color de la imagen. Los píxeles negros son
ahora blancos y viceversa.
        cvtColor(dst, cdst, CV_GRAY2BGR);

        vector<Vec4i> lines;

        //Aplicamos el detector de líneas a partir del detector de
bordes.
        //A mayor umbral, menos líneas se encuentra, menor
sensibilidad.
        // void HoughLinesP(InputArray image, OutputArray lines, double
rho, double theta, int threshold, double minLineLength=0, double
maxLineGap=0 )
        HoughLinesP(dst, lines, 1, CV_PI/180, UMBRAL, MIN_LENGTH,
MAX_GAP ); //HoughLinesP(dst, lines, 1, CV_PI/180, 50, 50, 10 );

        int contador = 0;
        //Conjunto de pares de puntos que representan las líneas.
        for(size_t i = 0; i < lines.size(); i++)
        {
                Vec4i l = lines[i];
                //Calculamos el tamaño de las líneas
                float dist = sqrt(pow((l[2]-l[0]),2)+pow((l[3]-
l[1]),2));
                line( cdst, Point(l[0], l[1]), Point(l[2], l[3]),
Scalar(0,0,255), 3, CV_AA);
                //cout << "Punto" << i+contador << ": " << Point(l[0],
l[1]) << endl;
                //cout << "Punto" << i+contador << "
transformado(yaml): " << Point(0-l[0], l[1]-sizeMapa.height) << endl;
                //cout << "Punto" << i+contador+1 << ": " <<
Point(l[2], l[3]) << endl;
                //cout << "Punto" << i+contador+1 << "
transformado(yaml): " << Point(0-l[2], l[3]-sizeMapa.height) << endl;

                //Obtenemos el origen del yaml respecto al segundo
sistema, transformado en píxeles:
                //Regla de tres: Si "resolución" metros es 1 píxel,
entonces "origen" es origenEnPixeles => origen/resolucion
                //Igual si se llegara a usar alguna vez el eje z
                float origenEnPixelX = origen.at(0)/resolucion;
                float origenEnPixelY = origen.at(1)/resolucion;

                float puntoPixelX1 = l[0]+origenEnPixelX;
                //cout << "¿Float convertido a int? " << puntoPixelX1
<< endl;
                float puntoPixelY1 = sizeMapa.height-
l[1]+origenEnPixelY;
                float puntoPixelX2 = l[2]+origenEnPixelX;
                float puntoPixelY2 = sizeMapa.height-
l[3]+origenEnPixelY;

                //HACER LA TRANSFORMADA FINAL
                //cout << "Punto" << i+contador << "
transformado(origen mapa): " << Point(puntoPixelX1, puntoPixelY1) <<
endl;
```

```
                //cout << "Punto" << i+contador+1 << "
transformado(origen mapa): " << Point(puntoPixelX2, puntoPixelY2) <<
endl;
                //Transformar de píxeles a metros
                //Regla de tres: Si 1 píxel es "resolución" en metros,
entonces puntoEnPixeles es puntoEnMetros

                //PUNTO DE INICIO DE LÍNEA EN METROS
                geometry_msgs::Point punto1;
                punto1.x = puntoPixelX1*resolucion;
                punto1.y = puntoPixelY1*resolucion;

                //      cout << "Punto" << i+contador << " de
geometry_msgs: " << punto1 << endl;

                //PUNTO DE FIN DE LÍNEA EN METROS
                geometry_msgs::Point punto2;
                punto2.x = puntoPixelX2*resolucion;
                punto2.y = puntoPixelY2*resolucion;

                //      cout << "Punto" << i+contador+1 << " de
geometry_msgs: " << punto2 << endl;

                //El par de puntos que representa el inicio y el fin de
la línea
                vector<geometry_msgs::Point> parPuntos;
                parPuntos.push_back(punto1);
                parPuntos.push_back(punto2);

                paresPuntos.push_back(parPuntos);

                contador++;
        }

        return paresPuntos;
}


void
ObjetivosEstanterias::establecerObjetivo(geometry_msgs::PoseStamped
objetivo)
{
        //Guardamos en el vector de objetivos para poder responder con
el servidor
        objx.push_back(objetivo.pose.position.x);
        objy.push_back(objetivo.pose.position.y);
        objyaw.push_back(tf::getYaw(objetivo.pose.orientation));

        //cout << "OBJETIVOYAW!!" <<
tf::getYaw(objetivo.pose.orientation) << endl;

        //24/03/17 Antes de mandar los objetivos, hay que quitar uno de
aquellos que estén muy cerca entre sí y con una orientación similar
        quitarCercanos();

        if(!quitar)
        {
                //Mostrar un objetivo con una flecha en RVIZ
                flechas.id = idFlechas;
                idFlechas += 2;
                flechas.pose.position.x = objetivo.pose.position.x;
```

```
                flechas.pose.position.y = objetivo.pose.position.y;
                flechas.pose.position.z = 0;
                flechas.pose.orientation.x = 0;
                flechas.pose.orientation.y = 0;
                flechas.pose.orientation.z =
objetivo.pose.orientation.z;
                flechas.pose.orientation.w =
objetivo.pose.orientation.w;

                markerArray.markers.push_back(flechas);
                pubMarkerArray.publish(markerArray);
        }

        quitar = false;
}

void ObjetivosEstanterias::pendienteCero(geometry_msgs::Point
pSiguiente)
{
        //t_perpendiculares = m = 0

        geometry_msgs::Point pS1;
        pS1.x = pSiguiente.x;
        pS1.y = pSiguiente.y - DIST_EST;
        pS1.z = 0;

        if(puntoValido(pS1))
        {
                objetivo.pose.position.x = pS1.x;
                objetivo.pose.position.y = pS1.y;
                objetivo.pose.orientation.z = sin((PI/2)/2);//1.0;//0.5
                objetivo.pose.orientation.w = cos((PI/2)/2);//0.0;//0.5

                establecerObjetivo(objetivo);
        }

        geometry_msgs::Point pS2;
        pS2.x = pSiguiente.x;
        pS2.y = pSiguiente.y + DIST_EST;
        pS2.z = 0;

        if(puntoValido(pS2))
        {

                objetivo.pose.position.x = pS2.x;
                objetivo.pose.position.y = pS2.y;
                objetivo.pose.orientation.z = sin(-(PI/2)/2);//-1.0;
                objetivo.pose.orientation.w = cos(-(PI/2)/2);//0.0;

                establecerObjetivo(objetivo);
        }
}

//Pendientes comprendidas entre -1 y 1
void ObjetivosEstanterias::pendienteProblematica(geometry_msgs::Point
pSiguiente, float m)
{
        //t perpendiculares = m

        geometry_msgs::Point pS1;
        pS1.x = pSiguiente.x + m;
```

```
        pS1.y = pSiguiente.y - (DIST_EST/m) * m;
        pS1.z = 0;

        if(puntoValido(pS1))
        {
                objetivo.pose.position.x = pS1.x;
                objetivo.pose.position.y = pS1.y;
                objetivo.pose.orientation.z = sin((atan(m) + PI/2) /
2);//sin((atan(-(1/m)) + PI) / 2);//0.0;
                objetivo.pose.orientation.w = cos((atan(m) + PI/2) /
2);//cos((atan(-(1/m)) + PI) / 2);//1.0;

                establecerObjetivo(objetivo);
        }

        geometry_msgs::Point pS2;
        pS2.x = pSiguiente.x - m;
        pS2.y = pSiguiente.y + (DIST_EST/m) * m;
        pS2.z = 0;

        if(puntoValido(pS2))
        {
                objetivo.pose.position.x = pS2.x;
                objetivo.pose.position.y = pS2.y;
                objetivo.pose.orientation.z = sin((atan(m) - PI/2) /
2);//sin(atan(-(1/m)) / 2);//0.0;
                objetivo.pose.orientation.w = cos((atan(m) - PI/2) /
2);//cos(atan(-(1/m)) / 2);//1.0;

                establecerObjetivo(objetivo);
        }
}

void ObjetivosEstanterias::pendienteInfinita(geometry_msgs::Point
pSiguiente)
{
        //t_perpendiculares = 1

        geometry_msgs::Point pS1;
        pS1.x = pSiguiente.x + DIST_EST;
        pS1.y = pSiguiente.y;
        pS1.z = 0;

        if(puntoValido(pS1))
        {
                objetivo.pose.position.x = pS1.x;
                objetivo.pose.position.y = pS1.y;
                objetivo.pose.orientation.z = sin(PI/2);//0.0;
                objetivo.pose.orientation.w = cos(PI/2);//-1.0;

                establecerObjetivo(objetivo);
        }

        geometry_msgs::Point pS2;
        pS2.x = pSiguiente.x - DIST_EST;
        pS2.y = pSiguiente.y;
        pS2.z = 0;

        if(puntoValido(pS2))
        {
                objetivo.pose.position.x = pS2.x;
```

```
                objetivo.pose.position.y = pS2.y;
                objetivo.pose.orientation.z = sin(0.0);//0 es igual que
sin(atan(0))
                objetivo.pose.orientation.w = cos(0.0);//1 es igual que
cos(atan(0))

                establecerObjetivo(objetivo);
        }
}

void ObjetivosEstanterias::pendienteNormal(geometry_msgs::Point
pSiguiente, float m)
{
        //t_perpendiculares = 1

        geometry_msgs::Point pS1;
        pS1.x = pSiguiente.x + DIST_EST;
        pS1.y = pSiguiente.y - (1/m) * DIST_EST;
        pS1.z = 0;

        if(puntoValido(pS1))
        {
                objetivo.pose.position.x = pS1.x;
                objetivo.pose.position.y = pS1.y;
                objetivo.pose.orientation.z = sin((atan(-(1/m)) + PI) /
2);//0.0;
                objetivo.pose.orientation.w = cos((atan(-(1/m)) + PI) /
2);//1.0;

                establecerObjetivo(objetivo);
        }

        geometry_msgs::Point pS2;
        pS2.x = pSiguiente.x - DIST_EST;
        pS2.y = pSiguiente.y + (1/m) * DIST_EST;
        pS2.z = 0;

        if(puntoValido(pS2))
        {
                objetivo.pose.position.x = pS2.x;
                objetivo.pose.position.y = pS2.y;
                objetivo.pose.orientation.z = sin(atan(-(1/m)) /
2);//0.0;
                objetivo.pose.orientation.w = cos(atan(-(1/m)) /
2);//1.0;

                establecerObjetivo(objetivo);
        }
}

// Si se quiere incluir el útimo punto, descomentar las siguientes
funciones miembro.
//void
ObjetivosEstanterias::pendienteCeroUltimoPunto(geometry_msgs::Point
p1)
//{
//      //t_perpendiculares = m = 0
//
//      geometry_msgs::Point pS1;
//      pS1.x = p1.x;
//      pS1.y = p1.y - 1;
```

```
//      pS1.z = 0;
//
//      if(puntoValido(pS1))
//      {
//              objetivo.pose.position.x = pS1.x;
//              objetivo.pose.position.y = pS1.y;
//              objetivo.pose.orientation.z = sin((PI/2)/2);//1.0;//0.5
//              objetivo.pose.orientation.w = cos((PI/2)/2);//0.0;//0.5
//
//              establecerObjetivo(objetivo);
//      }
//
//      geometry_msgs::Point pS2;
//      pS2.x = p1.x;
//      pS2.y = p1.y + 1;
//      pS2.z = 0;
//
//      if(puntoValido(pS2))
//      {
//              objetivo.pose.position.x = pS2.x;
//              objetivo.pose.position.y = pS2.y;
//              objetivo.pose.orientation.z = sin(-(PI/2)/2);//-1.0;
//              objetivo.pose.orientation.w = cos(-(PI/2)/2);//0.0;
//
//              establecerObjetivo(objetivo);
//      }
//}
//
////Pendientes comprendidas entre -1 y 1
//void
ObjetivosEstanterias::pendienteProblemUltimoPunto(geometry_msgs::Point
p1, float m)
//{
//      //t_perpendiculares = m
//
//      geometry_msgs::Point pS1;
//      pS1.x = p1.x + m;
//      pS1.y = p1.y - (1/m) * m;
//      pS1.z = 0;
//
//      if(puntoValido(pS1))
//      {
//              objetivo.pose.position.x = pS1.x;
//              objetivo.pose.position.y = pS1.y;
//              objetivo.pose.orientation.z = sin((atan(m) + PI/2) /
2);//sin((atan(-(1/m)) + PI) / 2);//0.0;
//              objetivo.pose.orientation.w = cos((atan(m) + PI/2) /
2);//cos((atan(-(1/m)) + PI) / 2);//1.0;
//
//              establecerObjetivo(objetivo);
//      }
//
//      geometry_msgs::Point pS2;
//      pS2.x = p1.x - m;
//      pS2.y = p1.y + (1/m) * m;
//      pS2.z = 0;
//
//      if(puntoValido(pS2))
//      {
//              objetivo.pose.position.x = pS2.x;
//              objetivo.pose.position.y = pS2.y;
```

```
//              objetivo.pose.orientation.z = sin((atan(m) - PI/2) /
2);//sin(atan(-(1/m)) / 2);//0.0;
//              objetivo.pose.orientation.w = cos((atan(m) - PI/2) /
2);//cos(atan(-(1/m)) / 2);//1.0;
//
//              establecerObjetivo(objetivo);
//      }
//}
//
//void
ObjetivosEstanterias::pendienteInfUltimoPunto(geometry_msgs::Point p1)
//{
//      //t_perpendiculares = 1
//
//      geometry_msgs::Point pS1;
//      pS1.x = p1.x + 1;
//      pS1.y = p1.y;
//      pS1.z = 0;
//
//      if(puntoValido(pS1))
//      {
//              objetivo.pose.position.x = pS1.x;
//              objetivo.pose.position.y = pS1.y;
//              objetivo.pose.orientation.z = sin(PI/2);//0.0;
//              objetivo.pose.orientation.w = cos(PI/2);//-1.0;
//
//              establecerObjetivo(objetivo);
//      }
//
//      geometry_msgs::Point pS2;
//      pS2.x = p1.x - 1;
//      pS2.y = p1.y;
//      pS2.z = 0;
//
//      if(puntoValido(pS2))
//      {
//              objetivo.pose.position.x = pS2.x;
//              objetivo.pose.position.y = pS2.y;
//              objetivo.pose.orientation.z = 0.0;
//              objetivo.pose.orientation.w = 1.0;
//
//              establecerObjetivo(objetivo);
//      }
//}
//
//void
ObjetivosEstanterias::pendienteNormalUltimoPunto(geometry_msgs::Point
p1, float m)
//{
//      //t_perpendiculares = 1
//
//      geometry_msgs::Point pS1;
//      pS1.x = p1.x + 1;
//      pS1.y = p1.y - (1/m) * 1;
//      pS1.z = 0;
//
//      if(puntoValido(pS1))
//      {
//              objetivo.pose.position.x = pS1.x;
//              objetivo.pose.position.y = pS1.y;
```

```
//              objetivo.pose.orientation.z = sin((atan(-(1/m)) + PI) /
2);//0.0;
//              objetivo.pose.orientation.w = cos((atan(-(1/m)) + PI) /
2);//1.0;
//
//              establecerObjetivo(objetivo);
//      }
//
//      geometry_msgs::Point pS2;
//      pS2.x = p1.x - 1;
//      pS2.y = p1.y + (1/m) * 1;
//      pS2.z = 0;
//
//      if(puntoValido(pS2))
//      {
//              objetivo.pose.position.x = pS2.x;
//              objetivo.pose.position.y = pS2.y;
//              objetivo.pose.orientation.z = sin(atan(-(1/m)) /
2);//0.0;
//              objetivo.pose.orientation.w = cos(atan(-(1/m)) /
2);//1.0;
//
//              establecerObjetivo(objetivo);
//      }
//}

//COMPROBACIÓN DE OBJETIVOS: DEVUELVE TRUE SI EL OBJETIVO ESTÁ DENTRO
DE LA ZONA DEL MAPA QUE EL ROBOT PUEDE RECORRER.
bool ObjetivosEstanterias::puntoValido(geometry_msgs::Point punto)
{
        //cout << "Comprobando validez del punto (" << "\n" << punto <<
")" << endl;
        bool resultado = true;

        //24/03/17 Primero comprobamos si hay una posible ruta al
objetivo
        nav_msgs::GetPlan ruta;
        tf::Quaternion quat;
        geometry_msgs::Quaternion gquat;
        float distanciaObjetivo;

        quat.setRPY(0.0,0.0,0.0);

        tf::quaternionTFToMsg(quat,gquat);

        //Preparamos la llamada al servidor de calculo de distancia
        ruta.request.goal.header.frame_id = "map";
        ruta.request.goal.pose.position.x = punto.x;
        ruta.request.goal.pose.position.y = punto.y;
        ruta.request.goal.pose.orientation = gquat;

        if(makePlan.call(ruta))
                distanciaObjetivo = ruta.response.plan.poses.size();
//Guardamos la distancia al  objetivo
        else
                ROS_ERROR("Error al calcular la distancia al objetivo
en nav_est");

        //Si no hay plan de ruta al objetivo (distanciaObjetivo = 0)
        if(distanciaObjetivo == 0)
        {
```

```
                //cout << "El punto " << "\n" << punto << "\n es
descartado" << endl;
                resultado = false;
        }
        else
        {
                //Paso 1: Hacer las transformaciones de sistemas de
coordenadas. Transformar el punto pasado.
                float sizeMapaY = sizeMapa.height * resolucion;
                geometry_msgs::Point puntoTransformado;
                puntoTransformado.x = punto.x - origen.at(0);
                puntoTransformado.y = origen.at(1) - punto.y +
sizeMapaY;

                //cout << "puntoTransformado(" << "\n" <<
puntoTransformado << ")" << endl;

                //Paso 2: Convertir el punto en píxel
                //Regla de tres: Si "resolución" metros es 1 píxel,
entonces "puntoTransformado" es puntoEnPixeles => origen/resolucion
                //Igual si se llegara a usar alguna vez el eje z
                int puntoEnPixelesX = puntoTransformado.x/resolucion;
                int puntoEnPixelesY = puntoTransformado.y/resolucion;

                int colorPixel =
(int)src.at<uchar>(puntoEnPixelesY,puntoEnPixelesX);
                //cout << colorPixel << endl;
                //Paso 3: Comprobar si el píxel es negro, gris o blanco
(if pixel.color < 255, entonces descartado)
                //También si el píxel se sale de la imagen, no
contarlo.
                if(colorPixel < 254 || puntoEnPixelesX > sizeMapa.width
|| puntoEnPixelesX < 0 || puntoEnPixelesY > sizeMapa.height ||
puntoEnPixelesY < 0)
                {
                        //cout << "El punto (" << puntoTransformado.x
<< ", " << puntoTransformado.y << ") en metros y (" << puntoEnPixelesX
<< ", " << puntoEnPixelesY << ") en pixeles es descartado." << endl;
                        resultado = false;
                }
        }
        return resultado;
}

void ObjetivosEstanterias::quitarCercanos()
{
        std::vector<float> objxAux = objx;
        std::vector<float> objyAux = objy;
        std::vector<float> objyawAux = objyaw;

        for(int i = 0; i < objx.size(); i++)
        {
                for(int j = 0; j < objx.size(); j++)
                {
                        //Comprobar que i y j no se pasan del tamaño de
objxAux.size()
                        if(i >= objxAux.size() || j >= objxAux.size())
                                break;
                        else
                        {
```

```
                                //Comprobar después que i y j son
diferentes
//                              if(i == j)
//                                      //ROS_WARN("No se puede comparar
un punto consigo mismo.");
//                              else
                                if(i != j)
                                {
                                        //Si están a unos 25 cm de
distancia o menos, tanto en el eje x como en el eje y, y la
orientación es parecida
                                        if(abs(objxAux.at(i)-
objxAux.at(j)) <= CERCANIA_OBJ && abs(objyAux.at(i)-objyAux.at(j)) <=
CERCANIA_OBJ &&
                                           abs(objyawAux.at(i)-
objyawAux.at(j)) <= 0.375)
                                        {
                                                cout << "Borrando " <<
objxAux.at(j) << ", " << objyAux.at(j) << endl;
                                                objxAux.erase
(objxAux.begin()+j);
                                                objyAux.erase
(objyAux.begin()+j);
                                                objyawAux.erase
(objyawAux.begin()+j);

                                                quitar = true;
                                        }
                                }
                        }
                }
        }

        objx = objxAux;
        objy = objyAux;
        objyaw = objyawAux;
}

void ObjetivosEstanterias::enviaRVIZ()
{
        //Necesitamos leer el mapa para poder descartar las posiciones
que están fuera del mismo (píxeles grises)
        //o que están en un obstáculo (píxeles negros)
        Mat src = imread(pgmPath, 0);

        geometry_msgs::Point p0;
        geometry_msgs::Point p1;

        float tam = 0.0; //= sqrt((pow(p1.x - p0.x, 2)) + (pow(p1.y -
p0.y, 2)));
        float m = 0.0; //= (p1.y - p0.y) / (p1.x - p0.x);
        int count = 0;

        // The line list needs two points for each line
        for(int v = 0; v < puntosEstanterias.size(); ++v)
        {
                line_list.id = idLineas;
                idLineas += 2;

                p0.x = puntosEstanterias.at(v).at(0).x;
                //      cout << "Coordenada X P0: " << p0.x << endl;
```

```
                p0.y = puntosEstanterias.at(v).at(0).y;

                line_list.points.push_back(p0);

                //              cout << "Punto" << v+count << " de
geometry_msgs: " << p0 << endl;

                p1.x = puntosEstanterias.at(v).at(1).x;
                p1.y = puntosEstanterias.at(v).at(1).y;

                line_list.points.push_back(p1);

                mArrayLines.markers.push_back(line_list);
                pubMArrayLines.publish(mArrayLines);

                //              cout << "Punto" << v+count+1 << " de
geometry_msgs: " << p1 << endl;

                tam = sqrt((pow(p1.x - p0.x, 2)) + (pow(p1.y - p0.y,
2)));
                m = (p1.y - p0.y) / (p1.x - p0.x);

                for (int i = 0; i <= tam / DIST; ++i)
                {
                        //No contamos el inicio de estantería
                        if(i != 0)
                        {
                                geometry_msgs::Point pSiguiente;
                                pSiguiente.x = (p0.x + i * ((p1.x -
p0.x)/(tam / DIST)));
                                pSiguiente.z = 0;

                                //La Z puede ir fuera del if.
                                //Hay que tener en cuenta la pendiente
infinita (la componente de los siguientes se obtiene sumando DIST)
                                if(isnan(m) || isinf(m))
                                  pSiguiente.y = (p0.y + i * DIST);
                                else
                                  pSiguiente.y = (p0.y + m * i * ((p1.x
- p0.x)/(tam / DIST)));

                                //cout << "Puntos intermedios de " <<
v+count << " y de " << v+count+1 << ": " << pSiguiente << endl;

                                //Rango de pendiente problemática
                                if(m > -1 && m < 1)
                                {
                                  //Con pendiente 0, nuevo conflicto
                                  if(m == 0)
                                          pendienteCero(pSiguiente);
                                  else

pendienteProblematica(pSiguiente, m);
                                }
                                else
                                {
                                  //Con pendiente infinita, nuevo
conflicto (en este caso, los puntos distan 1 metro de las rectas (p.X
= x +-1)
                                  if(isnan(m) || isinf(m))
                                          pendienteInfinita(pSiguiente);
```

```
                                        else
                                                pendienteNormal(pSiguiente,
m);
                                }
                        }
                }
// Si se quiere incluir el último punto de cada línea, descomentar el
siguiente código.
//              //Para crear los puntos de aproximación del último
punto (el final de la estantería)
//              //Rango de pendiente problemática
//              if(m > -1 && m < 1)
//              {
//                      //Con pendiente 0, nuevo conflicto
//                      if(m == 0)
//                              pendienteCeroUltimoPunto(p1);
//                      //Pendiente entre -1 y 1 distinta de 0
//                      else
//                              pendienteProblemUltimoPunto(p1, m);
//              }
//              //Si la pendiente es mayor que 1 o menor que -1
//              else
//              {
//                      //Con pendiente infinita, nuevo conflicto (en
este caso, los puntos distan 1 metro de las rectas (p.X = x +-1)
//                      if(isnan(m) || isinf(m))
//                              pendienteInfUltimoPunto(p1);
//                      //Si la pendiente es mayor que 1 o menor que -
1, pero no infinita
//                      else
//                              pendienteNormalUltimoPunto(p1, m);
//              }
                ++count;
        }
}

int main( int argc, char** argv )
{
  ros::init(argc, argv, "nav_est");
  ObjetivosEstanterias objetivosEstanterias;
  ros::spin();
}
```

7.2.7 Librería navegación estanterías

```
#include <cmath>

//Ros
#include <ros/ros.h>
#include <std_srvs/Trigger.h>
#include <etapa_guiado/Objetivo.h>
#include <visualization_msgs/Marker.h>
#include <visualization_msgs/MarkerArray.h>
#include <geometry_msgs/PoseStamped.h>
#include <geometry_msgs/Point.h>
#include <tf/transform_datatypes.h>
#include <nav2d_navigator/commands.h>
#include <localizacion_productos/ObjetivosLoc.h>
#include <localizacion_estanterias/ObjetivosEstanterias.h>
//c
#include <iostream>
#include <vector>
#include <math.h>
//Ficheros cabecera para la extracción del origen y la resolución en
el yaml
#include <fstream>
#include <boost/algorithm/string.hpp>
#include <sstream>

//Programa para la navegación de las estanterias
//opencv2
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"

//MOD 24/03/17 PARA CALCULAR LA DISTANCIA A LOS OBJETIVOS
#include <nav_msgs/GetPlan.h>

#define PI 3.14159265

//Definir la longitud de las divisiones de la estantería
#define DIST 0.80 //1.0 //1.5 para el pasillo
//Definir el umbral del detector de líneas
#define UMBRAL 40//70 para lab //50 //30
//Definir la longitud mínima para validar las líneas
#define MIN_LENGTH 75//25
//Definir la "distancia" máxima entre píxeles para considerarlos
enlazados.
#define MAX_GAP 15//10

//Distancia mínima entre objetivos
#define CERCANIA_OBJ 0.5 //0.25

//Definir la distancia que va a haber entre el robot y la estantería.
#define DIST_EST 0.6 //0.70 //0.5 //0.75 // 1.0

using namespace std;
using namespace cv;

class ObjetivosEstanterias
{
        public:
                ObjetivosEstanterias();
```

```
                void initMarkers();
                float resolucionMapa();
                string obtenerDatos(string strTemp);
                vector<float> creaOrigen(string cadena);
                vector<float> obtenerOrigen();
                vector<vector<geometry_msgs::Point> > estanterias();

                void pendienteCero(geometry_msgs::Point pSiguiente);
                void pendienteProblematica(geometry_msgs::Point
pSiguiente, float m);
                void pendienteInfinita(geometry_msgs::Point
pSiguiente);
                void pendienteNormal(geometry_msgs::Point pSiguiente,
float m);

                //Si se quiere incluir el último punto de cada línea,
descomentar estas funciones miembro.
//              void pendienteCeroUltimoPunto(geometry_msgs::Point p1);
//              void pendienteProblemUltimoPunto(geometry_msgs::Point
p1, float m);
//              void pendienteInfUltimoPunto(geometry_msgs::Point p1);
//              void pendienteNormalUltimoPunto(geometry_msgs::Point
p1, float m);

                void establecerObjetivo(geometry_msgs::PoseStamped
objetivo);

                bool puntoValido(geometry_msgs::Point punto);
                void enviaRVIZ();
                bool
mandarObjetivos(localizacion_estanterias::ObjetivosEstanterias::Reques
t  &req,

localizacion_estanterias::ObjetivosEstanterias::Response &res);

                void quitarCercanos();

                bool afs;

        private:
                vector<float> xyyaw;
                visualization_msgs::Marker points, line_strip,
line_list, flechas;
                visualization_msgs::MarkerArray markerArray,
mArrayLines;
                //Para contar la línea actual del fichero que se está
leyendo.
                int linea;
                char* userHome;
                char yamlPath[256];
                char pgmPath[256];
                bool objetivosPasados = false;
                //MOD para mandar los objetivos a
objetivos_localizacion.cpp
                geometry_msgs::PoseStamped objetivo;
                ros::Publisher goalPublisher;
                ros::Publisher marker_pub, pubMarkerArray,
pubMArrayLines;
                ros::ServiceServer server;
                ros::NodeHandle n, gp, so, s, nhArray, nhLines;
                ros::ServiceClient servicioObjetivos;
```

```
                vector<vector<geometry_msgs::Point> >
puntosEstanterias;

                std::vector<float> objx;
                std::vector<float> objy;
                std::vector<float> objyaw;

                vector<float> origen;
                float resolucion;

                //Imagen del mapa y tamaño de la misma en píxeles
                Mat src;
                Size sizeMapa;

                //Contadores para el id de las flechas y el de las
líneas. Contador es siempre par e idLineas es siempre impar
                int idFlechas, idLineas;

                //24/03/17
                ros::NodeHandle mp;
                ros::ServiceClient makePlan;

                bool quitar;
};
```

8. Referencias

http://doc.aldebaran.com/2-1/_downloads/nao_safetyguide_2014_en_fr_sp_pt_gr_it_nl.pdf

http://doc.aldebaran.com/2-1/nao/getting_out_of_the_box.html#setup-wizard-nao

http://doc.aldebaran.com/2-1/naoqi/index.html#naoqi-api

https://github.com/SeanXP/Nao-Robot/blob/master/python/socket/simple/client.py

https://docs.python.org/2/howto/sockets.html

https://stackoverflow.com/questions/33649368/how-to-post-json-as-request-body-with-jetty-http-client

http://www.ros.org/

http://wiki.ros.org/indigo/Installation/Ubuntu

http://wiki.ros.org/ROS/Tutorials

Programming Robots with ROS, Morgan Quigley, Brian Gerkey & William D. Smart.

Learning ROS for Robotics Programming – First Edition.

Learning ROS for Robotics Programming – Second Edition.

Robot Operating System (ROS), The complete reference (Volume 1).

Think Python, O'Reilly, Allen B. Downey.