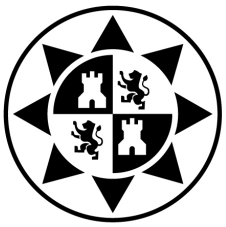


Universidad
Politécnica
de Cartagena

industriales
etsii UPCT

ROS: Robot Operating System

Titulación: I.T.I Electrónica Industrial
Intensificación:
Alumno/a: Alvaro García Cazorla
Director/a/s: Jose Luis Muñoz Lozano

Cartagena, 7 de Septiembre de 2013

1. Memoria de proyecto

El objetivo de este proyecto fin de carrera es el de presentar una visión general de ROS, sin entrar en detalle dentro de lo que es su programación, pero mostrando todos los pasos necesarios a realizar para poder trabajar con el sistema en los equipos compatibles, además de mostrar como funciona su estructura y unos primeros pasos necesarios para la correcta comprensión de ROS, además se dará una visión objetiva de sus ventajas e inconvenientes respecto a los demás sistemas existentes en la actualidad, y de sus posibilidades de futuro.

Todas las imágenes que se pueden ver en este documento son propias a excepción de las que se indique lo contrario.

2. ROS

Para poder sacar el máximo rendimiento a ROS, primero tenemos que comprender en que consiste el sistema, y cuales son sus principales ventajas respecto al resto de sistemas que se pueden encontrar hoy en día.

2.1 Introducción a ROS

2.1.1 Los robots de antes y los de ahora

Un robot es una máquina controlada por ordenador y programada para moverse, manipular objetos y realizar trabajos a la vez que interacciona con su entorno. Su objetivo principal es el de sustituir al ser humano en tareas repetitivas, difíciles, desagradables e incluso peligrosas de una forma más segura, rápida y precisa. Algunas definiciones aceptadas son las siguientes:

"Dispositivo multifuncional reprogramable diseñado para manipular y/o transportar material a través de movimientos programados para la realización de tareas variadas." (*Robot Institute of America, 1979*).

“Máquina o ingenio electrónico programable, capaz de manipular objetos y realizar operaciones antes reservadas solo a las personas.” *(Diccionario de la lengua española, 2013).*

Los robots exhiben tres elementos claves según la definición adoptada:

- Programable, lo que significa disponer de capacidades computacionales y de manipulación de símbolos (el robot es un computador).

- Capacidad mecánica, que lo capacita para realizar acciones en su entorno y no ser un mero procesador de datos (el robot es una máquina).

- Flexibilidad, puesto que el robot puede operar según un amplio rango de programas y manipular material de formas distintas.

Con todo, se puede considerar un robot como una máquina complementada con un computador o como un computador con dispositivos de entrada y salida sofisticados.

La idea más ampliamente aceptada de robot está asociada a la existencia de un dispositivo de control digital que, mediante la ejecución de un programa almacenado en memoria, va dirigiendo los movimientos de un brazo o sistema mecánico. El cambio de tarea a realizar se verifica ordenando el cambio de programa.

El concepto de máquinas automatizadas se remonta a la antigüedad, con mitos de seres mecánicos vivientes. Los autómatas, o máquinas semejantes a personas, ya aparecían en los relojes de las iglesias medievales, y los relojeros del siglo XVIII eran famosos por sus ingeniosas criaturas mecánicas.

El control por realimentación, el desarrollo de herramientas especializadas y la división del trabajo en tareas más pequeñas que pudieran realizar obreros o máquinas fueron ingredientes esenciales en la automatización de las fábricas en el siglo XVIII. A medida que mejoraba la tecnología se desarrollaron máquinas especializadas para tareas como poner tapones a las botellas o verter caucho líquido en moldes para neumáticos. Sin embargo, ninguna de estas máquinas tenía la versatilidad del brazo humano, y no podían alcanzar objetos alejados y colocarlos en la posición deseada.

En la década de 1890 el científico Nikola Tesla, inventor, entre muchos otros dispositivos, de los motores de inducción, ya construía vehículos controlados a distancia por radio.

Las máquinas más próximas a lo que hoy en día se entiende como robots fueron los "teleoperadores", utilizados en la industria nuclear para la manipulación de sustancias radiactivas. Básicamente se trataba de servomecanismos que, mediante sistemas mecánicos, repetían las operaciones que simultáneamente estaba realizando un operador.

Inmediatamente después de la Segunda Guerra Mundial comienzan los primeros trabajos que llevan a los robots industriales. A finales de los 40 se inician programas de investigación en los laboratorios de *Oak Ridge y Argonne National Laboratories* para desarrollar manipuladores mecánicos para elementos radiactivos. Estos manipuladores eran del tipo "maestro-esclavo", diseñados para que reprodujeran fielmente los movimientos de brazos y manos realizados por un operario.

El inventor estadounidense George C. Devol desarrolló en 1954 un *dispositivo de transferencia programada articulada* (según su propia definición); un brazo primitivo que se podía programar para realizar tareas específicas.

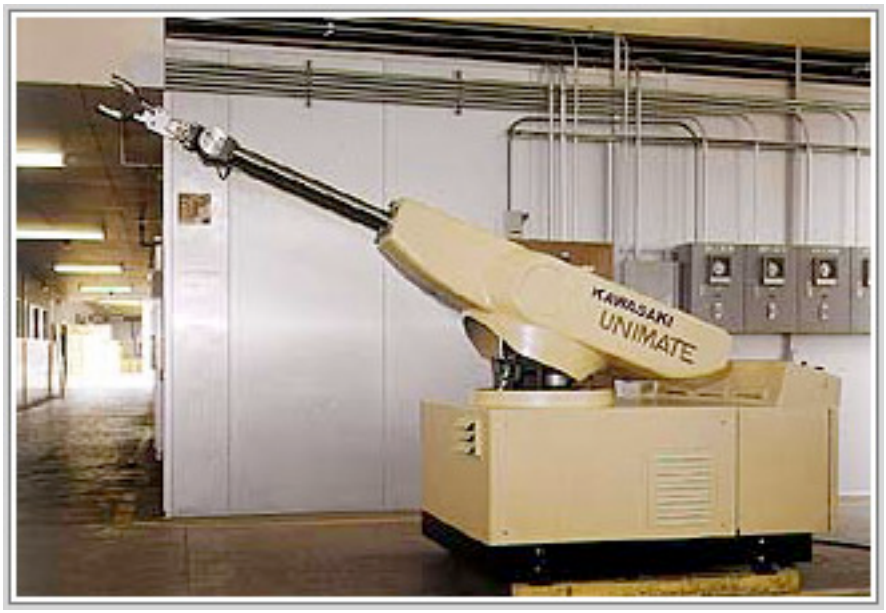
En 1958, Devol se unió a Joseph F. Engelberger y, en el garaje de este último, construyeron un robot al que llamaron *Unimate.* Era un dispositivo que utilizaba un computador junto con un manipulador que conformaban una "máquina" que podía ser "enseñada" para la realización de tareas variadas de forma automática. En 1962, el primer *Unimate* fue instalado a modo de prueba en una planta de la *General Motors* para funciones de manipulación de piezas y ensamblaje, con lo que pasó a convertirse en el primer robot *industrial*. Devol y Engelberger fundarían más tarde la primera compañía dedicada expresamente a fabricar robots, *Unimation, Inc.*, abreviación de *Universal Automation*

Se puede considerar este punto como el inicio de la era de la Robótica tal como la conocemos, mediante la utilización de los robots programados, una nueva y potente herramienta de fabricación.

Durante la década de los 60, un nuevo concepto surge en relación con los anteriores avances. En vistas a una mayor flexibilidad, se hace necesaria la realimentación sensorial. En 1962, Tomovic y Boni desarrollan una mano con un sensor de presión para la detección del objeto que proporcionaba una señal de realimentación al motor.

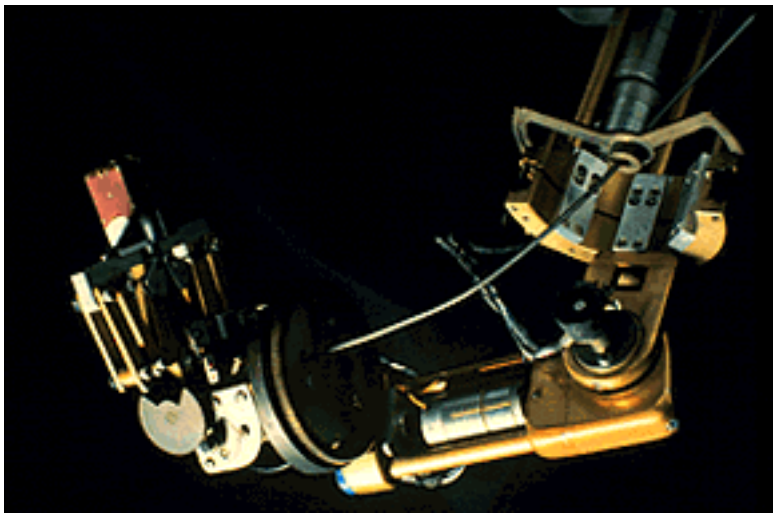
En 1963 se introduce el robot comercial *VERSATRAN* por la *American Machine and Foundry Company (AMF)*. Y ese mismo año se desarrollan otros brazos manipuladores como el *Roehampton* y el *Edinburgh.*

En 1967 y 1968 *Unimation* recibe sus primeros pedidos para instalar varios robots de la serie *Unimate 2000* en las cadenas de montaje de la *General Motors*. Al año siguiente los robots ensamblaban todos los coches *Chevrolet Vega* de esta compañía.



robot Unimate 2000 (1967)
www.motorshopdemammoet.com

En 1968 se publica el desarrollo de un computador con "manos", "ojos" y "oídos" (manipuladores, cámaras de TV y micrófonos) por parte de McCarthy en el *Stanford Artificial Intelligence Laboratory.* En el mismo año, Pieper estudia el problema cinemático de un manipulador controlado por un computador. También este año, la compañía japonesa *Kawasaki Heavy Industries* negocia con *Unimation* la licencia de sus robots. Este momento marca el inicio de la investigación y difusión de los robots industriales en Japón.



El brazo *Stanford*(1969).

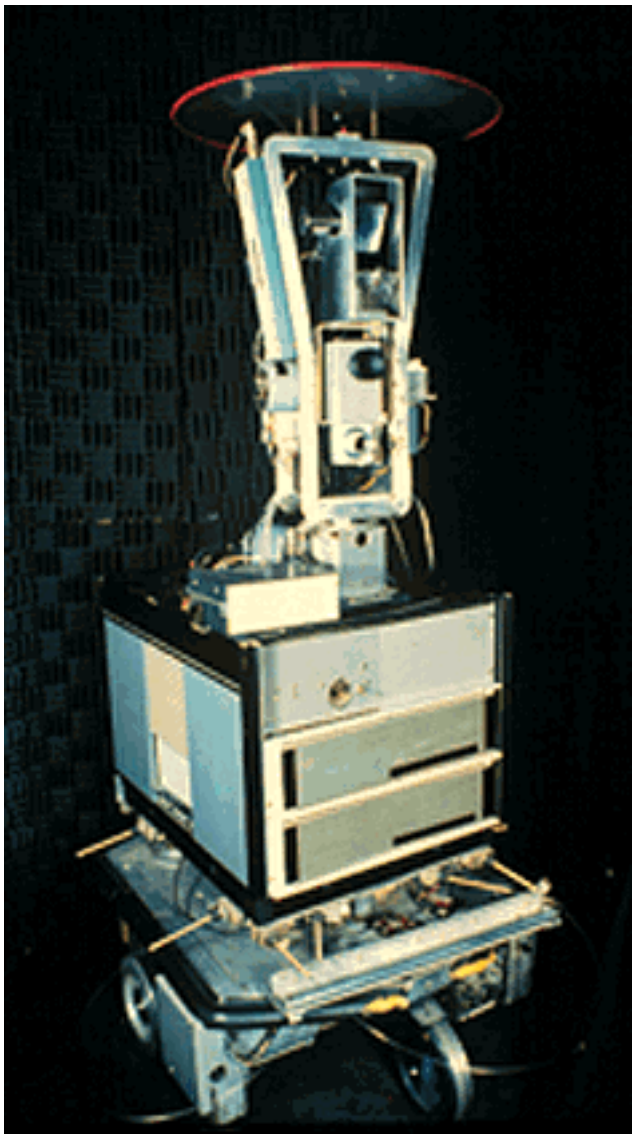
Wikipedia

Las primeras aplicaciones industriales en Europa, aplicaciones de robots industriales en cadenas de fabricación de automóviles, datan de los años 1970 y 1971. En este último año, Kahn y Roth analizan el comportamiento dinámico y el control de un brazo manipulador.

Durante la década de los 70, la investigación en robótica se centra en gran parte en el uso de sensores externos para su utilización en tareas de manipulación. Es también en estos años cuando se consolida definitivamente la presencia de robots en las cadenas de montaje y plantas industriales en el ámbito mundial.

En 1972 se desarrolló en la universidad de *Nottingham*, Inglaterra, el *SIRCH*, un robot capaz de reconocer y orientar objetos en dos dimensiones. Este mismo año, la empresa japonesa *Kawasaki* instala su primera cadena de montaje automatizada en *Nissan*, Japón, usando robots suministrados por *Unimation, Inc*.

En 1973, Bolles y Paul utilizan realimentación visual en el brazo *Stanford* para el montaje de bombas de agua de automóvil. También este mismo año, la compañía sueca *ASEA* (futura *ABB*), lanza al mercado su familia de robots *IRB 6* e *IRB 60*, para funciones de perforación de piezas, y la empresa KUKA Robot Group lanzó al mercado su robot Famulus, que era el primer robot con seis ejes electromecánicos.



Shakey, (1970)
Wikipedia

También este mismo año, la empresa *Cincinatti Milacron* introduce el T3 (*The Tomorrow Tool*), su primer robot industrial controlado por computador. Este manipulador podía levantar más de 100 libras y seguir objetos móviles en una línea de montaje.

En 1975, Will y Grossman, en *IBM*, desarrollaron un manipulador controlado por computador con sensores de contacto y fuerza para montajes mecánicos. Este mismo año, el ingeniero mecánico estadounidense Victor Scheinman, cuando estudiaba la carrera en la Universidad de *Stanford*, California, desarrolló un manipulador polivalente realmente flexible conocido como *Brazo Manipulador Universal Programable* (*PUMA*, siglas en inglés). El *PUMA* era capaz de mover un objeto y colocarlo en cualquier orientación en un lugar deseado que estuviera a su alcance. El concepto básico multiarticulado del *PUMA* es la base de la mayoría de los robots actuales.

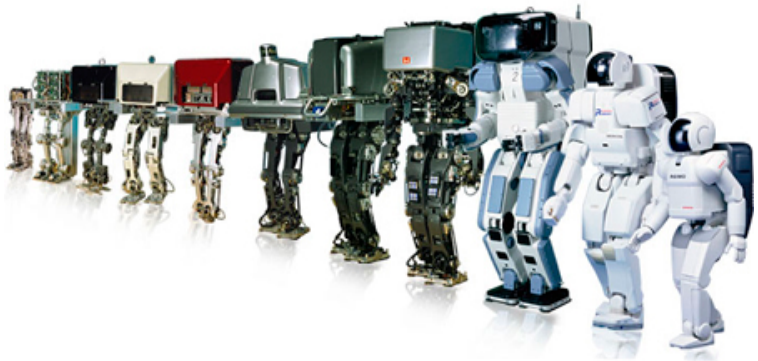
En 1976, estudios sobre el control dinámico llevados a cabo en los laboratorios *Draper*, Cambridge, permiten a los robots alinear piezas con movimientos laterales y rotacionales a la vez.

En 1979 Japón introduce el robot SCARA (Selective Compliance Assembly Robot Arm), y la compañía italiana DEA (Digital Electric Automation), desarrolla el robot PRAGMA para la General Motors.

El robot PUMA.

En 1982, el robot *Pedesco*, se usa para limpiar un derrame de combustible en una central nuclear. También se pone un gran énfasis en los campos de visión artificial, sensorización táctil y lenguajes de programación. Gracias a los primeros pasos dados por compañías como *IBM* o *Intelledex Corporation*, que introdujo en 1984 el modelo ligero de ensamblaje 695, basado en el microprocesador *Intel* 8087 y con software *Robot Basic*, una modificación del *Microsoft Basic*, actualmente se tiende al uso de una interfaz (el ordenador) y diversos lenguajes de programación especialmente diseñados, que evitan el "cuello de botella" que se producía con la programación "clásica". Esta puede ser ahora *on-line* u *off-line*, con interfaces gráficas (*user-friendly interfaces*) que facilitan la programación, y un soporte *SW+HW* que tiende a ser cada vez más versátil.

En el año 2000 se presenta en sociedad al robot ASIMO un robot humanoide capaz de desplazarse de forma bípeda e interactuar con las personas, creado por la empresa Honda Motor Co. Ltd, que tras varias actualizaciones en al año 2011 consiguió funcionar a una velocidad de 9 km/h, diferenciar entre 3 personas hablando a la vez y completar funciones tales como coger un vaso de papel y llenarlo de agua sin derramar ni una sola gota.



Evolución de los robots humanoides (2013)
gizmologia.com

2.1.2 Qué es ROS

ROS (Robot Operating System) es una plataforma de desarrollo open source para sistemas robóticos. Proporciona toda una serie de servicios y librerías que simplifican considerablemente la creación de aplicaciones complejas para robots.

Es similar a otras plataformas de desarrollo para robots existentes en la actualidad, y su mayor virtud, es la de haber sabido aunar lo mejor de cada uno de estos sistema, juntando todo en un solo sistema capaz de comunicarse tanto con los robots mas modernos, como con los ya existentes en el mercado.

Desde su creación, ROS se ha diseñado para facilitar el intercambio de software entre los aficionados y los profesionales de la robótica en todo el mundo debido a su enfoque didáctico y abierto, lo que ha permitido la construcción de una gran comunidad de colaboradores a lo largo de todo el mundo.

A la hora de desarrollar, ROS permite el uso de distintos lenguajes de programación. De forma oficial soportan Python, C++ y Lisp además de muchas otras como java (todavía en fase experimental pero apoyada por google), Lua, etc.

ROS puede ser ejecutado sobre maquinas tipo Unix, principalmente Ubuntu y Mac OS X aunque por parte de la comunidad puede encontrarse soporte para otras plataformas como Fedora, Gentoo, etc

ROS cuenta con una enorme comunidad de desarrollo, que se compone de investigadores, aficionados, fabricantes de hardware y empresas que le dan soporte como Willow Garage, o incluso Google.

De esa forma ROS ya dispone de soporte nativo para un gran número de hardware como Nao[1], Lego Mindstorms[2], las aspiradoras Roomba[3], o incluso robots de investigación valorados en miles de dolares como el PR-2[4]

Robot PR-2
Willow Garage

[1] http://www.aldebaran-robotics.com/en/

[2] http://www.lego.com/es-es

[3] http://www.irobot.com/global/es/home.aspx

[4] http://www.willowgarage.com/pages/pr2/overview

Todo esto, sumado a la gran cantidad de ejemplos, librerias listas para funcionar (con algoritmos de navegación, visión artificial, etc) y la creciente comunidad ROS es un gran punto de partida para iniciar el camino en el apasionante mundo de la Robotica.

2.1.3 Objetivos de ROS

El objetivo de ROS no es ser un sistema pionero en la robótica, el objetivo principal de ROS es apoyar el código reutilizable en la investigación robótica y el desarrollo. El sistema es una estructura distribuida en procesos (también conocidos como *nodos o clases*) que permite a los ejecutables ser diseñados de forma individual y utilizarse de forma flexible en tiempo real. Estos procesos se pueden agrupar en *paquetes* y *pilas* , que pueden ser fácilmente compartidos y distribuidos. ROS también es compatible con el sistema de repositorios, que permiten la colaboración a nivel internacional permitiendo tomar decisiones independientes sobre el desarrollo y la ejecución, pero con la ventaja de poder ayudarse de las herramientas o archivos creados por otros en sus proyectos.

El objetivo principal de este proyecto es el de compartir y colaborar, pero hay varios objetivos mas dentro del marco ROS:

- Sencillez: ROS está diseñado para ser tan sencillo como sea posible de modo que el código escrito para ROS se pueda utilizar con otros marcos robot de software. ROS es fácil de integrar con otros sistemas operativos de robots y ya ha sido integrado con OpenRAVE, Orocos y Player.
- Modelo de bibliotecas: es el modelo de desarrollo preferido, consiste en escribir una serie de programas con interfaces funcionales y limpias que permitan una rápida modificación.
- Independencia de idiomas: el marco ROS es fácil de implementar en cualquier lenguaje de programación moderno. Se ha implementado en Python, C++, y LISP; y actualmente existen bibliotecas experimentales en Java y Lua.
- Modo test: ROS tiene una orden llamada ROSTEST que hace que sea fácil acceder a un modo prueba de sistema.
- Escala: ROS es adecuada para sistemas grandes y ejecución de los procesos de gran tamaño.

2.1.4 Sistemas operativos admitidos

ROS en la actualidad sólo se ejecuta de forma totalmente funcional en plataformas basadas en UNIX. el Software de ROS está principalmente probado en Ubuntu y Mac OS X, aunque la comunidad ROS ha contribuido al apoyo a Fedora, Gentoo, Arch Linux y además de otras plataformas Linux.
Respecto a Microsoft Windows, es un sistema en el que podremos instalar nuestro sistema ROS, aunque todavía existen algunos pequeños errores que se están intentando solucionar.

2.1.5 Nuevas bibliotecas

Tanto el núcleo del sistema ROS, como los útiles, herramientas y bibliotecas son creados y actualizados regularmente como una distribución de ROS . Esta distribución es similar a

una distribución de Linux y ofrece un conjunto de software compatible para que todo el mundo pueda usarla, además de que por su naturaleza Open Source te permite modificar cualquier distribución para poder adaptarla a tus necesidades.

2.1.6 Contribuyendo

Como ROS es un sistema de código abierto, cualquier persona puede contribuir tanto con el sistema como con las bibliotecas que son compatibles disponiendo para ello de una sección específica dentro de la propia pagina web del software[5].

2.2 Funcionamiento del sistema

ROS tiene tres niveles de conceptos: el nivel del sistema de archivos, el nivel de Computación Gráfica, y el nivel comunitario. Estos niveles y conceptos se resumen a continuación

2.2.1 Sistema de archivos

Son recursos que se encuentran en el propio programa:

- Paquetes: Los paquetes son la unidad principal para organizar software en ROS. Un paquete puede contener procesos ejecutables (*nodos*), una biblioteca dependiente, conjuntos de datos, archivos de configuración, o cualquier otra cosa que sea útil para una organización conjunta.
- Manifiestos: proporcionan metadatos sobre un paquete, incluyendo su información de licencia y dependencias, así como información específica del compilador.
- Pilas: Es una colección de paquetes que tienen una misma función.
- Manifiestos de pilas: proporcionan datos sobre una pila, incluyendo su información de licencia y sus dependencias en otras pilas.
- Mensajes: definen las estructuras de datos para los mensajes enviados en ROS.
- Servicios: definen la solicitud y estructuras de datos de respuesta de los servicios requeridos por ROS.

2.2.2 Computación a nivel Gráfico

La computación a nivel gráfico es la red ROS que se encarga de procesar todos los datos. Los conceptos básicos son *nodos* , *maestro* , *mensajes* y temas, los cuales proporcionan los datos de diferentes maneras:

- Nodos: Los nodos son procesos que llevan a cabo cálculos. ROS está diseñado para ser modular en una escala básica. Un sistema de control de robot comprenderá usualmente muchos nodos. Por ejemplo, un nodo controla un telémetro láser, un nodo controla los motores de las ruedas, un nodo realiza localización, un nodo realiza la planificación de ruta, un nodo proporciona una vista gráfica del sistema, y así sucesivamente.

---

[5] http://wiki.ros.org/Contributing

- Maestro: El Maestro proporciona registro de nombres y la búsqueda para el resto de la Computación Gráfica. Sin el Maestro, los nodos no serían capaces de encontrar mensajes entre sí, intercambiar, o invocar los servicios.
- Mensajes : Los nodos se comunican entre sí pasando mensajes. Un mensaje es simplemente una estructura de datos que comprende los tipos de campos. Los mensajes pueden incluir estructuras arbitrariamente anidadas y matrices (al igual que las estructuras de C).
- Temas : Los mensajes se enrutan a través de un sistema de transporte de publicación / suscripción semántica. Un nodo envía un mensaje por *publicar* a un determinado tema. El tema es un nombre que se utiliza para identificar el contenido del mensaje. Un nodo que está interesado en un determinado tipo de datos se *suscribe* al tema correspondiente. Puede haber varios editores y suscriptores concurrentes a un mismo tema, y un único nodo puede publicar y / o suscribirse a múltiples temas. En general, los editores y suscriptores no son conscientes de la existencia de los demás. Se puede pensar en un tema como un Bus de mensajes. Cada Bus tiene un nombre, y cualquier persona puede conectarse al bus para enviar o recibir mensajes, siempre y cuando sean del tipo correcto.

2.2.3 La comunidad y ROS

Los conceptos comunitarios ROS Nivel ROS son recursos que permiten a las comunidades el intercambio del software y del conocimiento. Estos recursos incluyen:

- Distribución: Son colecciones de versionadas en pilas que se pueden instalar. Las distribuciones juegan un papel similar al de las distribuciones de Linux: hacen que sea más fácil para instalar un conjunto de programas informáticos, y también mantener versiones consistentes a través de un conjunto de software.
- Repositorios: ROS se basa en una red federada de repositorios de código, donde diferentes instituciones pueden desarrollar y lanzar sus propios componentes de software del robot.
- El wiki de ROS[6]: La comunidad Wiki es el foro principal para documentar la información sobre ROS. Cualquier persona puede inscribirse con una cuenta y contribuir con su propia documentación, facilitar las correcciones o actualizaciones, escribir tutoriales y más.
- Respuestas de ROS: un sitio de preguntas y respuestas para contestar a las preguntas relacionadas con ROS.
- Blog: El blog de Willow Garage(creador de ROS) ofrece actualizaciones periódicas, incluyendo fotos y videos.

3.Instalación del sistema

En este capítulo vamos a comenzar con la toma de contacto con el programa, y para ello vamos a enumerar los pasos necesarios para poder usarlo en algunos de los sistemas operativos soportados, aunque en nuestro caso se detallará más Ubuntu ya que es con el que se ha realizado este documento.

---

[6] http://wiki.ros.org

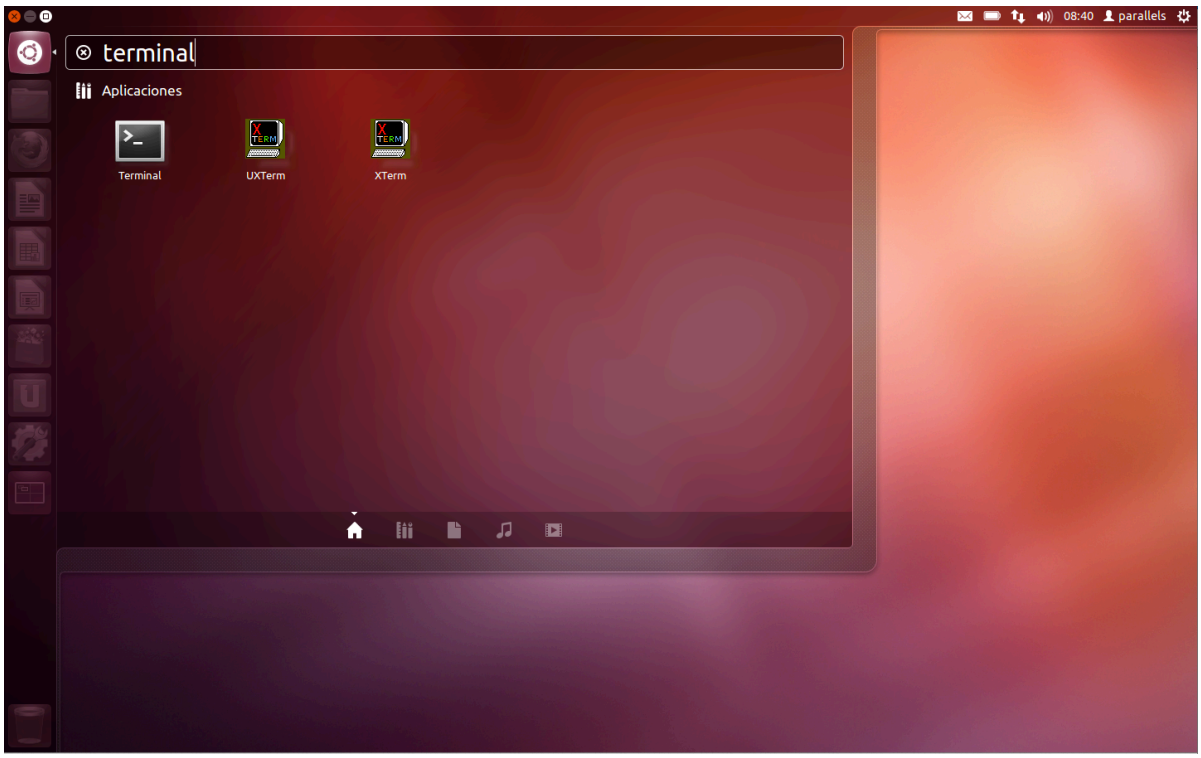
3.1 Sistemas Linux

Vamos a proceder a instalar ROS en Linux, en nuestro caso se ha escogido Ubuntu 12.04 ya que es la ultima actualización estable disponible en el momento de realizar este documento.

Primero comprobamos que tengamos actualizado nuestro sistema, en esta versión podemos encontrar en la parte izquierda de la pantalla principal el programa llamado “update manager”.

Vista principal de Ubuntu 12.04

Para comenzar abrimos la utilidad de Terminal.



Vista de uno de los accesos a terminal.

A continuación preparamos el sistema para que acepte los paquetes de información de la pagina del repositorio packages.ros.org, para ello usamos la función correspondiente a nuestro sistema.

$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu precise main" > /etc/apt/sources.list.d/ros-latest.list'

Tras introducir la función anterior, nos pedirá la contraseña de usuario para confirmar que queremos aceptar esta función, y terminal volverá a modo espera.

Ventana de terminal tras preparar el sistema.

Lo siguiente es descargarte el fichero correspondiente a tu teclado, para ello usamos la función:

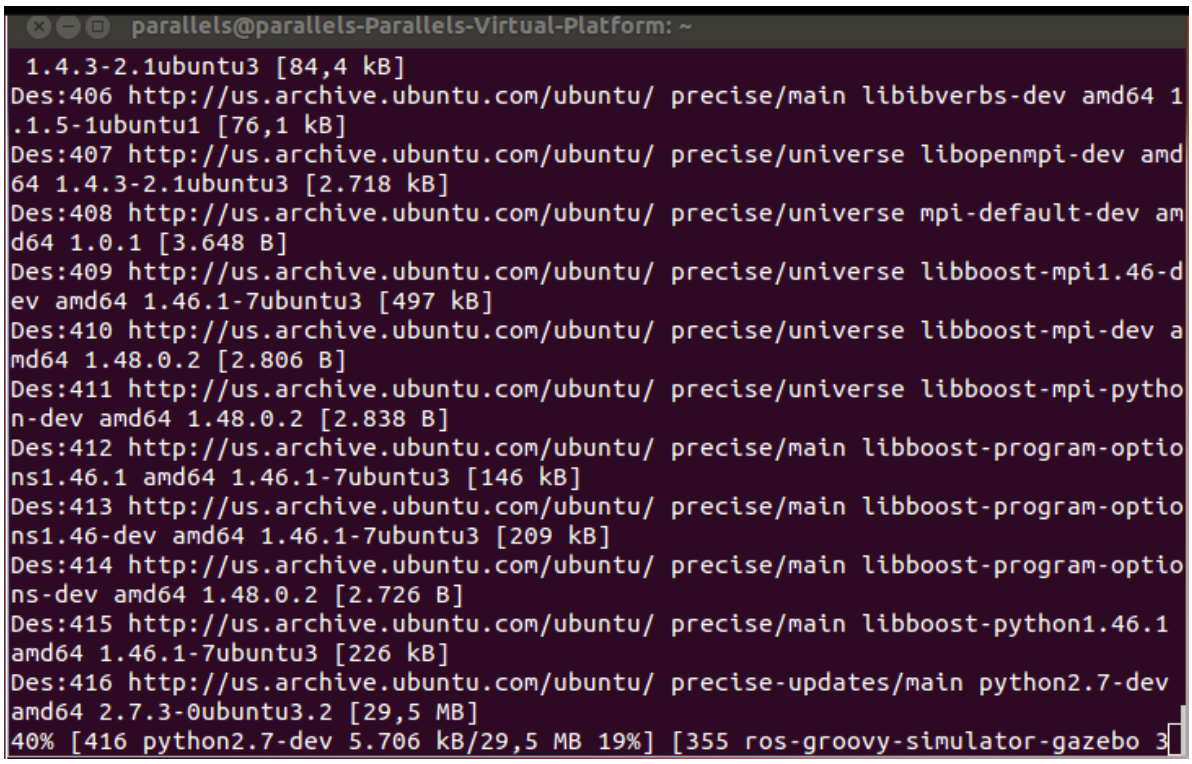
$ wget http://packages.ros.org/ros.key -O - | sudo apt-key add -

Ventana de terminal tras instalar el teclado de ROS.

Una vez preparado nuestro sistema para instalar ROS procedemos a la instalación del sistema operativo, para ello usamos la función que nos descargará el sistema por completo (en el caso de tener algun tipo de problema con la instalación de alguna fase se puede instalar uno por uno todos los componentes necesarios).

Primero comprobamos que esté todo actualizado con la función

$ sudo apt-get update



Ventana de ejemplo del uso de update.

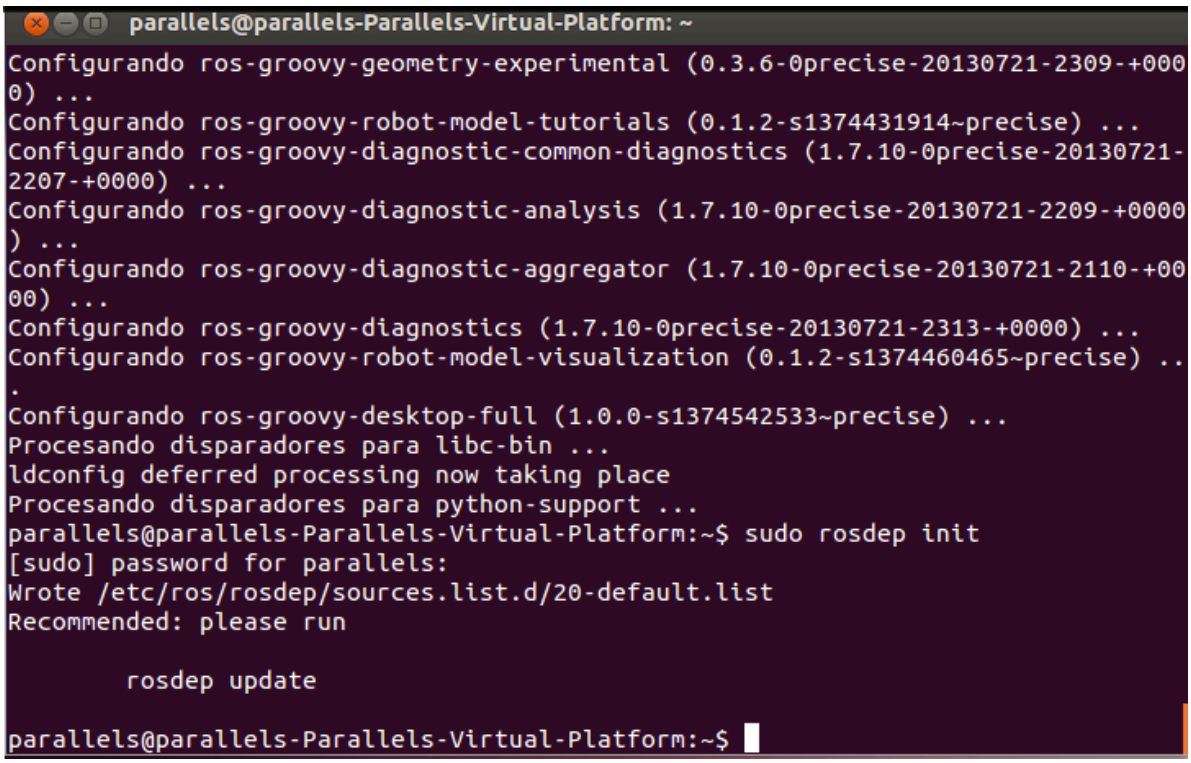
y luego usamos la función:

$ sudo apt-get install ros-groovy-desktop-full

y nada mas pulsar la tecla intro el programa se pondrá a funcionar descargando todo lo necesario para que el sistema funcione.

Durante la instalación podremos configurar varios aspectos que tendrá nuestro sistema, como por ejemplo si solo va a trabajar en modo local, o se podrá utilizar también en remoto.

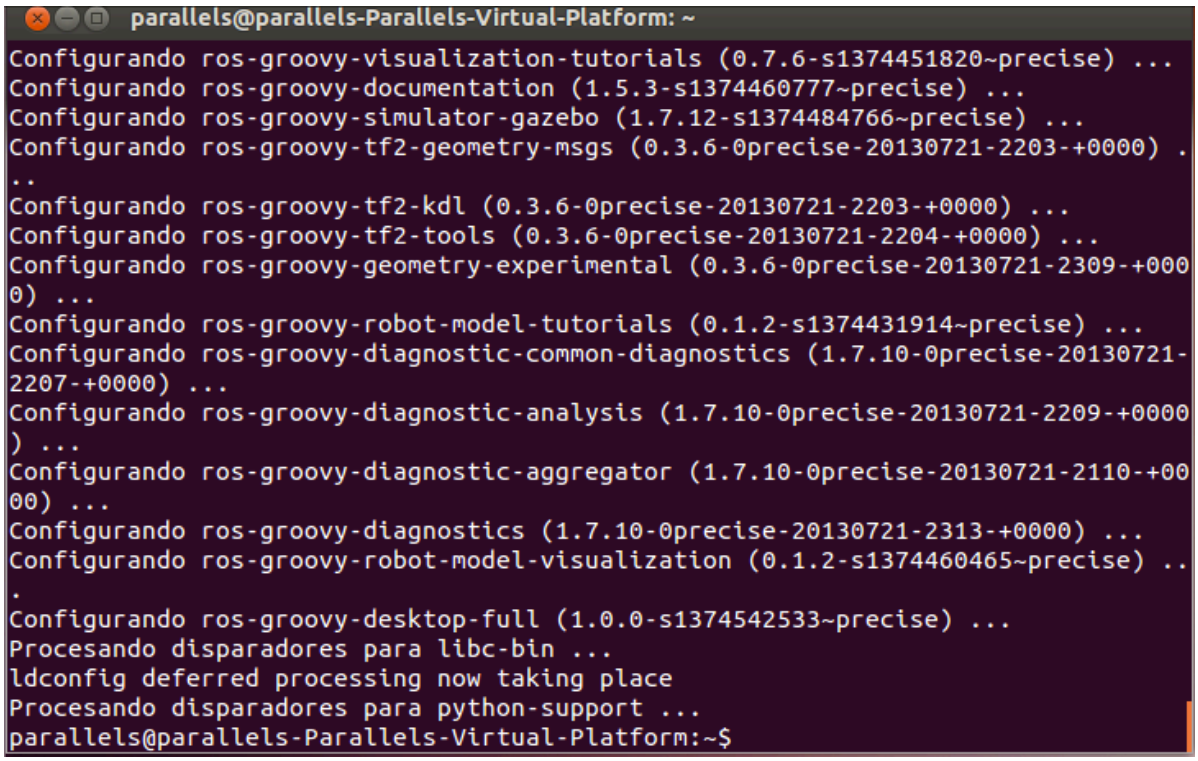
Cuando termine de descargar e instalar todos los paquetes deberemos ver algo parecido a esto:



Ventana de terminal tras la instalación de ROS.

A continuación iniciamos el programa rosdep y comprobamos que esté actualizado, para ello usamos primero la función:

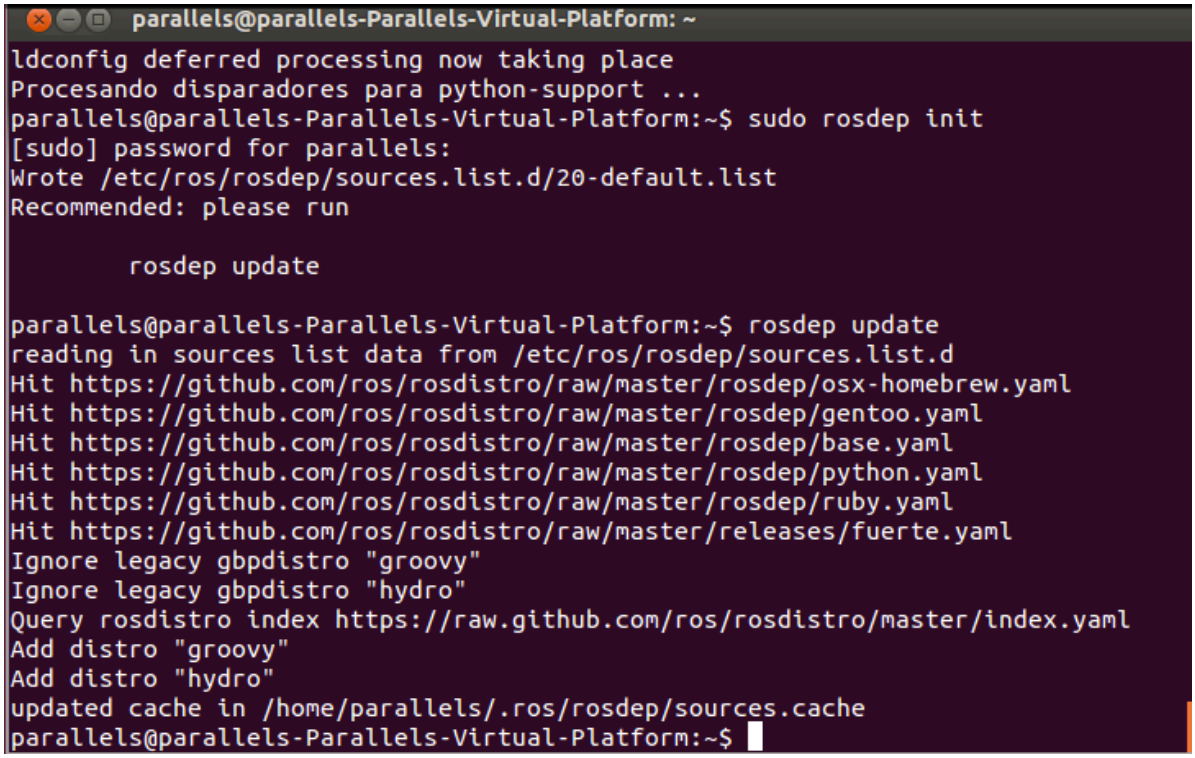
$ sudo rosdep init



Ventana de terminal tras inicializar Rosdep.

Y luego la función:

$ rosdep update



Ventana de terminal tras actualizar Rosdep.

Ya tenemos ROS instalado en nuestro sistema Ubuntu, y ahora solo nos quedaría añadir un par de configuraciones mas que son recomendables.

La primera es para que las variables de entorno que creemos se añadan automaticamente a nuestra sesión, para ello usamos la función:

$ echo “source /opt/ros/groovy/setup.bash” >> ~/.bashrc

Y luego:

$ source ~/.bashrc

También es conveniente instalar rosinstall, que es un añadido que nos permitirá descargarnos facilmente el codigo fuente de muchos añadidos con solo un comando, para ello usamos el comando:

$ sudo apt-get install python-rosinstall

Tras esto ya tendremos nuestro sistema listo para trabajar con el.

3.2 Sistemas MAC OS X

Para proceder a instalar ROS en MAC OS X, primero deberemos descargarnos un programa denominado Homebrew que nos permitirá y ayudará a instalar todo lo necesario para poder trabajar con él.

La aplicación hombrew de instala escribiendo en el terminal de MAC OS X el siguiente código:

ruby -e "$(curl -fsSL https://raw.github.com/mxcl/homebrew/go)"

tras indicarle al programa que estamos seguros que queremos instalar el programa, se descargará e instalará todo los componentes del programa

De forma adicional podemos ejecutar desde terminal el codigo

brew doctor

Y el programa comprobará los archivos que va a necesitar, si están instalados o desactualizados.

Lo siguiente que debemos hacer es instalar el software adicional necesario

$ brew update

$ brew install cmake

y ahora comenzamos a descargar los primeros ficheros necesarios para ROS, en este caso la distribución Hydro.

$ brew tap ros/hydro

$ brew tap Homebrew/science

Ahora necesitaremos añadia a nuestro fichero /.bashrc lo necesario para que hombrew trabaje de forma conjunta con terminal

export PATH=/usr/local/bin:/usr/local/share/python:$PATH

export PYTHONPATH=" /usr/local/python2.7 /site-packages:$PYTHONPATH"

Y ahora configuramos el terminal para que cada vez que lo abramos no tengamos que reintroducir esta ultima parte del código.

$ brew untap ros/DISTRO

Ahora prepararemos nuestro sistema para poder descargar los ultimos ficheros necesarios para usar ROS

$ sudo easy_install pip

$ sudo pip install -U wstool rosdep rosinstall rosinstall_generator rospkg catkin-pkg Distribute

$ sudo rosdep init

$ rosdep update

Ahora vamos a crear un espacio de trabajo para catkin.

$ mkdir ~/ros_catkin_ws

$ cd ~/ros_catkin_ws

Y ahora vamos a a instalar las librerias o complementos necesarios para ROS

$ rosinstall_generator desktop-full --rosdistro hydro --deps --wer-only > hydro-desktop-full-wet.rosinstall

$ wstool init -j8 src hydro-desktop-wet.rosinstall

También podemos descargar lo basico para el uso usando los dos codigos anteriores, pero solo poniendo desktop en lugar de desktop-full.

Ahora procedemos a comprobar que tenemos todas las dependencias requeridas.

$ rosdep install --from-paths src --ignore-src --rosdistro hydro -y

Y ya tendremos nuestro sistema preparado para poder usar ROS.

3.3 Sistemas Windows

Este es posiblemente el mas sencillo de todos los sistemas para instalar ROS, aunque no se debe olvidar que los ficheros para trabajar con Windows están en fase Beta, y no están exentos de fallos de programación, y desde la propia página de Willow Garage nos invitan a ayudar a mejorarlo.

Para proceder a la instalación solo deberemos ir a la página

http://wiki.ros.org/win_ros/hydro/Msvc%20SDK

descargarnos y ejecutar los ficheros necesarios para su correcto funcionamiento.

4. Primeros pasos con ROS

4.1 Iniciar/configurar el programa

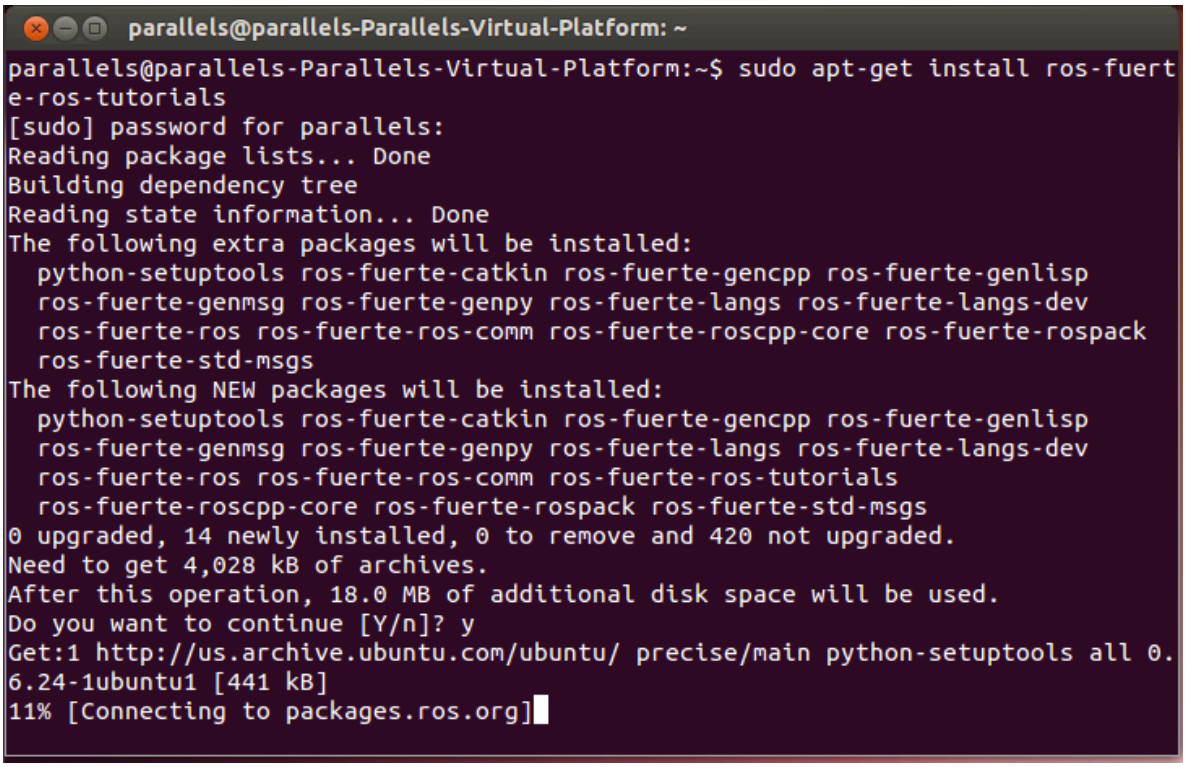
Una vez que tenemos el sistema en nuestro ordenador, vamos a proceder a mostrar el funcionamiento básico de ROS para mas adelante poder comprender mas a fondo el sistema a nivel jerárquico.

En este caso para probar que la configuración del programa sea la correcta vamos a utilizar uno de los programas ya creados y que podemos encontrar dentro de la pagina de ROS, hablamos de turtlesim, que es un sencillo programa que utilizaremos de aqui en adelante para nuestro ejemplos.

Para descargarnos esta distribución a nuestro sistema utilizaremos el comando

$ sudo apt-get install ros-fuerte-ros-tutorials

Y tras introducir nuestra contraseña de administrador nos debería aparecer una pantalla como esta:



Ventana de terminal mientras se instala los tutoriales.

Cuando llegue al 100% terminal volverá al modo de espera y ya tendremos instalado nuestro ejemplo.

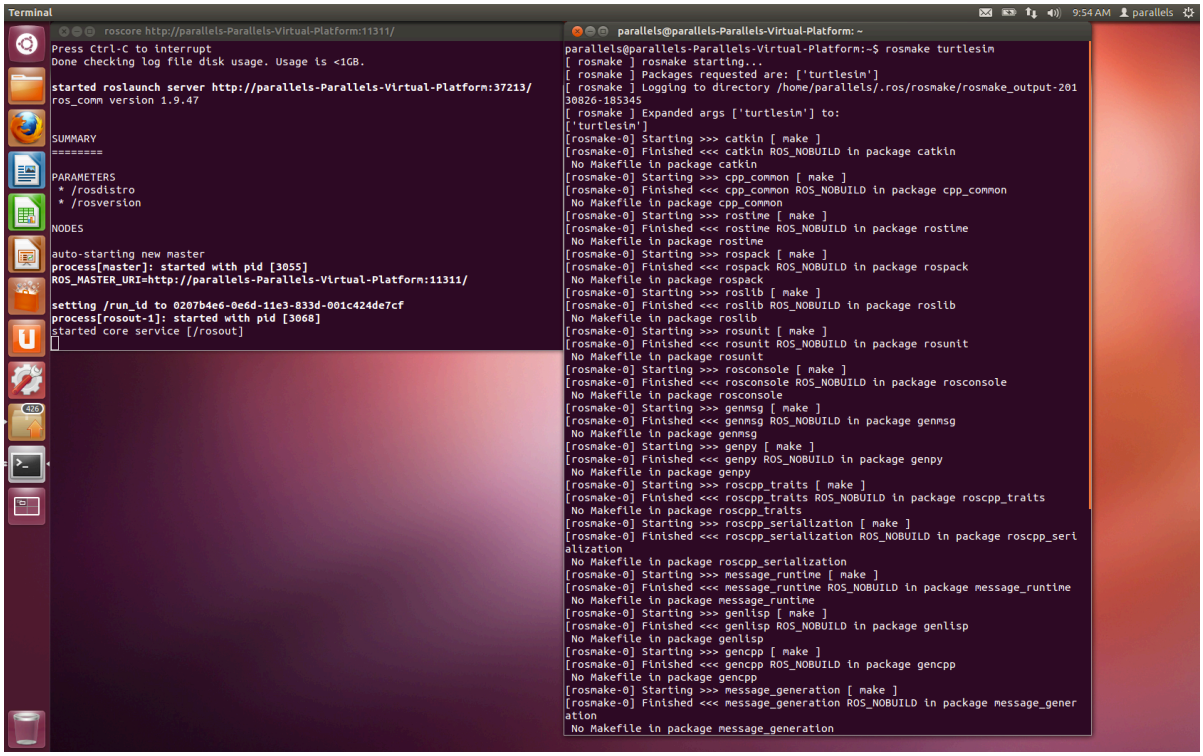
4.2 Ejecutar un programa

Para ejecutar nuestro programa de prueba tendremos que abrir una nueva ventana de terminal y teclear

$ rosmake turtlesim

tras lo que el programa nos generará todo el directorio turtlesim con todo lo necesario para trabajar con él

También señalar que este paso solo es necesario la primera vez que ejecutamos el programa, las demás veces podremos obviar este paso ya que en nuestro sistema ya tendremos todos los ficheros generados y solo habrá que activarlos.

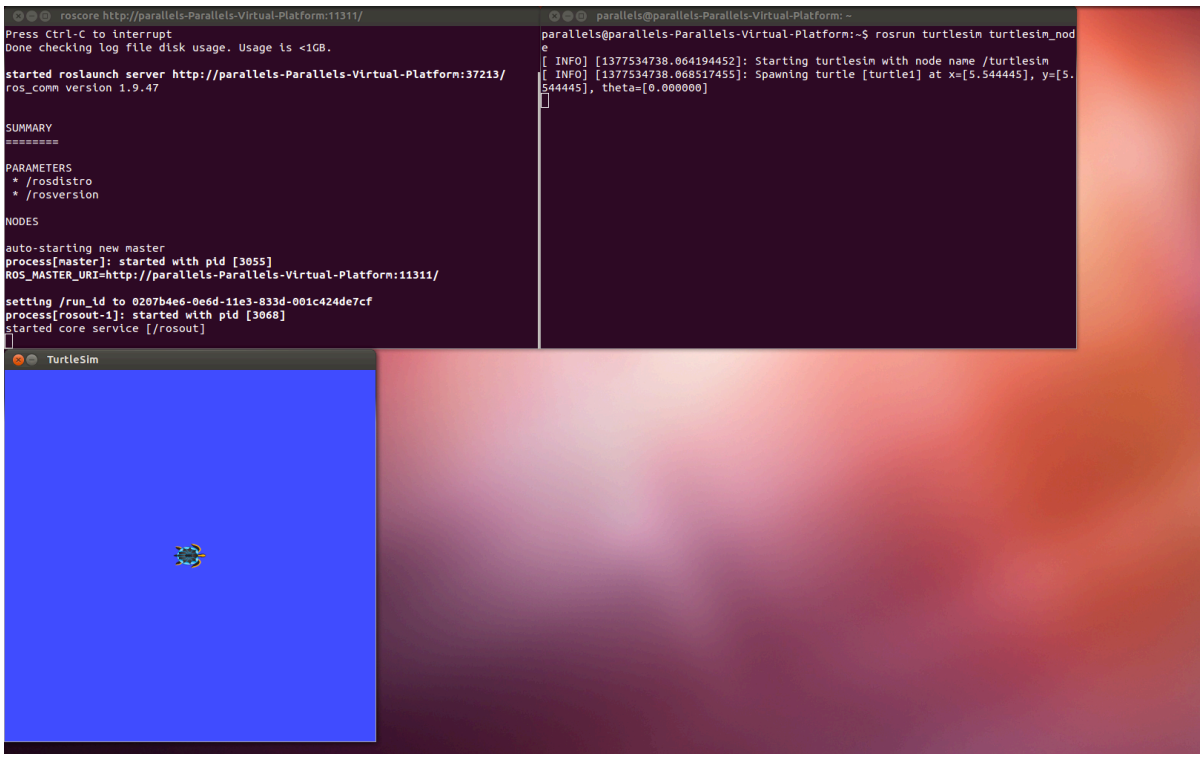


Ventana de terminal de ROS generando el paquete turtlesim.

Luego solo tendremos que teclear

$ rosrun turtlesim turtlesim_node

y con esto veremos algo parecido a esto:



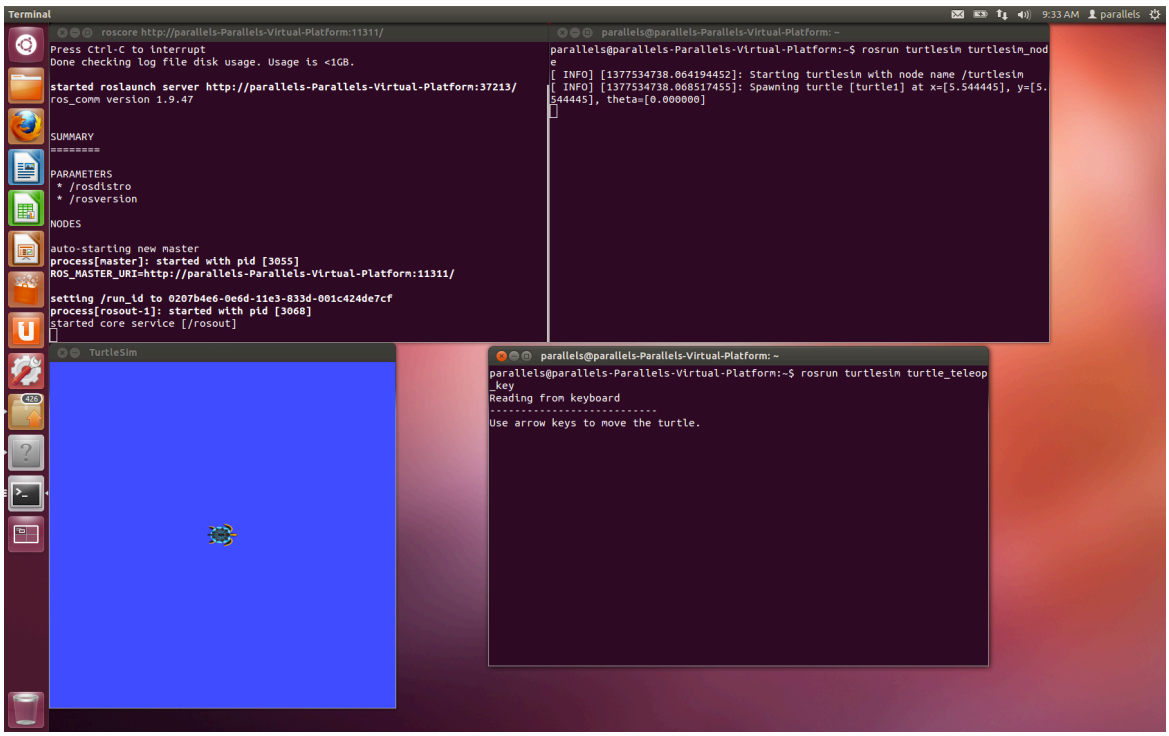
Ventana de terminal con turtlesim ejecutándose.

4.3 Ejecutar un subprograma

Ahora vamos a proceder a interactuar con nuestro programa, para ello vamos a activar un subprograma que nos va a permitir manejar con el teclado la tortuga que vemos en nuestra pantalla.

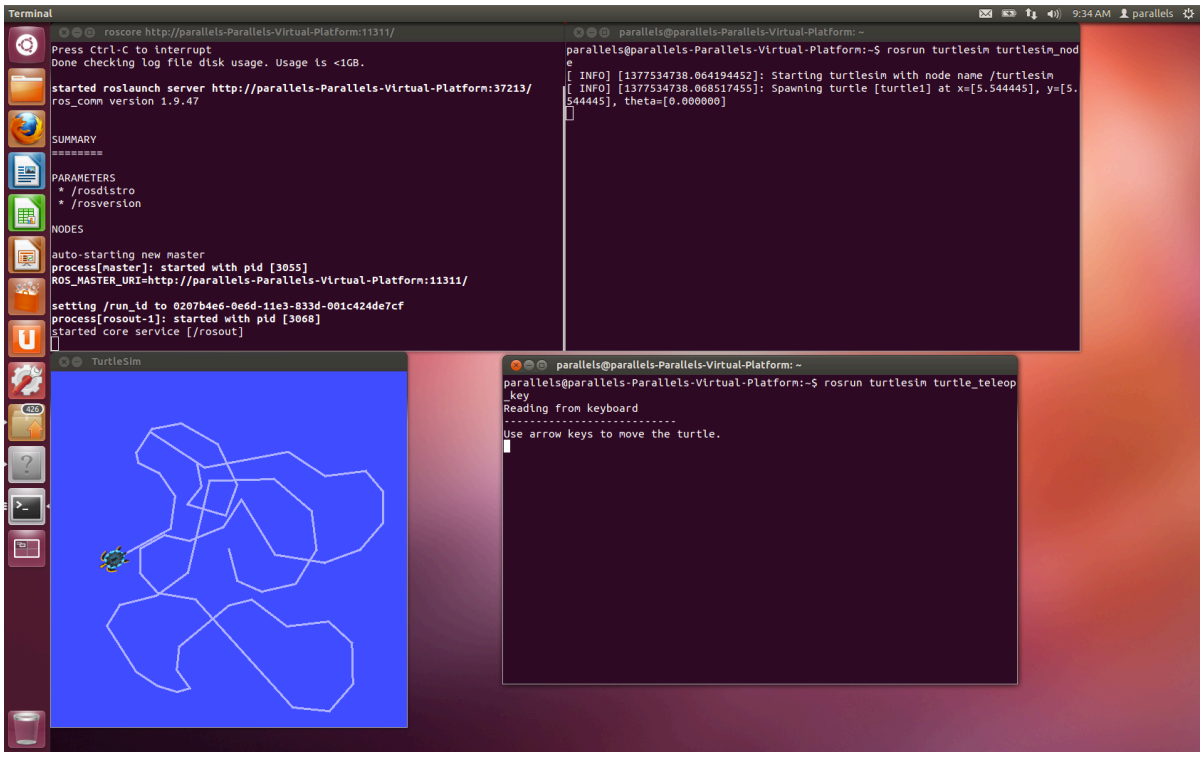
Para ello abrimos una nueva ventana de terminal e introducimos:

$ rosrun turtlesim turtle_teleop_key



Ventana de terminal que muestra la pantalla de teleoperación con teclado.

Ahora ya puedes mover con las teclas de dirección la tortuga del ejemplo



Ventana que muestra el funcionamiento de turtlesim tras pulsar las teclas de dirección.

5. ROS en profundidad

Tras una breve explicación vamos a proceder a una explicación mas en profundidad de como funciona ROS, su jerarquía y su forma de trabajar.

5.1 Estructura

Lo primero que vamos a ver es la estructura del sistema operativo, de lo mas global (repositorio) a lo mas especifico (nodo).

5.1.1 Repositorio

Son ficheros compuestos por uno o mas paquetes y pilas, y que nos permiten descargarnos los ficheros necesarios de forma mas cómoda.

5.1.2 Pila (Stack)

Es una colección de paquetes que comparten una misma funcionalidad, seria el equivalente a una librería en otros sistemas de programación, y dentro de estos podemos encontrar unos ficheros con metadatos que nos proporcionan la información para que estos funcionen de forma correcta (dependencias, compilación...)

5.1.3 Paquete

Es la unidad principal de organización en ROS, contiene todo lo necesario para hacer ser funcional y es análogo a un paquete en C.

5.1.4 Nodo

Son los procesos ejecutable que están incluidos dentro de los paquetes.

Normalmente de utilizan varios nodos en los programas.

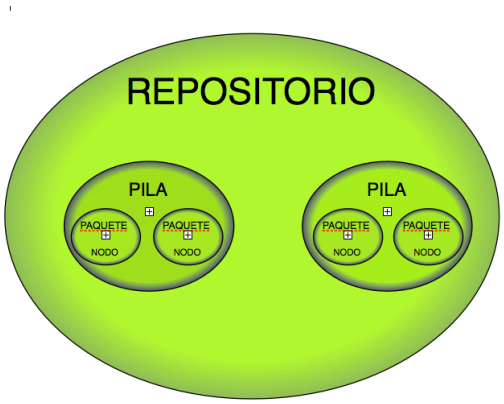


Imagen esquemática de la jerarquización de ROS.

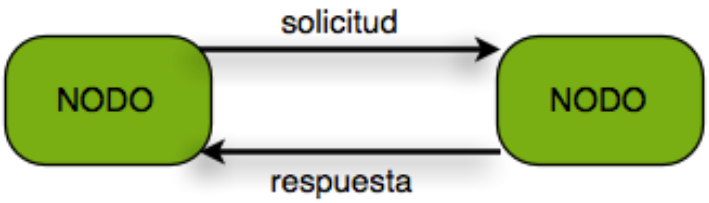
5.2 Computación en ROS

5.2.1 Servicios

Es el tipo de arquitectura encargada de realizar la comunicación entre nodos, utilizan dos tipos de mensajes, uno para la solicitud, y otro que es la respuesta dada por el otro nodo a esa petición.

En los programas podremos encontrar nodos servidor y nodos clientes.

Tras realizar la solicitud, el nodo servidor se queda en modo espera hasta recibir respuesta por parte del nodo cliente, y en el caso de que sea el cliente el que realiza una petición, el nodo servidor la procesará y responderá al cliente con la información requerida



Mecanismo de comunicación entre nodos.

5.2.2 Tópicos

Tópicos o temas, son los nombres que identifican el contenido de un mensaje; y estos se enrutan de dos formas, una publicador y otra suscriptor.

Un nodo que está interesado en un determinado tipo de datos se *suscribe* al tema correspondiente.

Puede haber varios editores y suscriptores concurrentes a un mismo tema, y un único nodo puede publicar y / o suscribirse a múltiples temas. En general, los editores y suscriptores no son conscientes de la existencia de los demás.

Se puede pensar en un tema como un Bus de mensajes. Cada Bus tiene un nombre, y cualquier persona puede conectarse al bus para enviar o recibir mensajes, siempre y cuando sean del tipo correcto.

5.2.3 Mensajes

Los nodos se comunican entre sí pasando mensajes. Un mensaje es simplemente una estructura de datos, que comprende los tipos de campos. Los mensajes pueden incluir estructuras arbitrariamente anidadas y matrices (al igual que las estructuras de C).

5.2.4 Maestro

El Maestro proporciona registro de nombres y la búsqueda para el resto de los nodos. Sin el Maestro, estos no serían capaces de encontrar mensajes entre sí, intercambiar, o invocar los servicios, lo que hace que sea totalmente indispensable a la hora de ejecutar cualquier tipo de programa.

5.2.5 Bags

Las bolsas son un formato para guardar y reproducir datos de un mensaje de ROS, permitiéndonos almacenar una serie de ordenes y después repetirlas secuencialmente. Las bolsas son un mecanismo importante para el almacenamiento de datos, tales como datos de un sensor, que puede ser difícil de recoger, pero es necesaria para desarrollar y probar algoritmos.

6. El desarrollo de un programa

6.1 Espacio de trabajo

Como en cualquier otro programa, en nuestro sistema necesitaremos un lugar donde trabajar, en este caso se requiere de un área para crear nuestras pilas y paquetes, y su posterior modificación en caso de ser necesario.

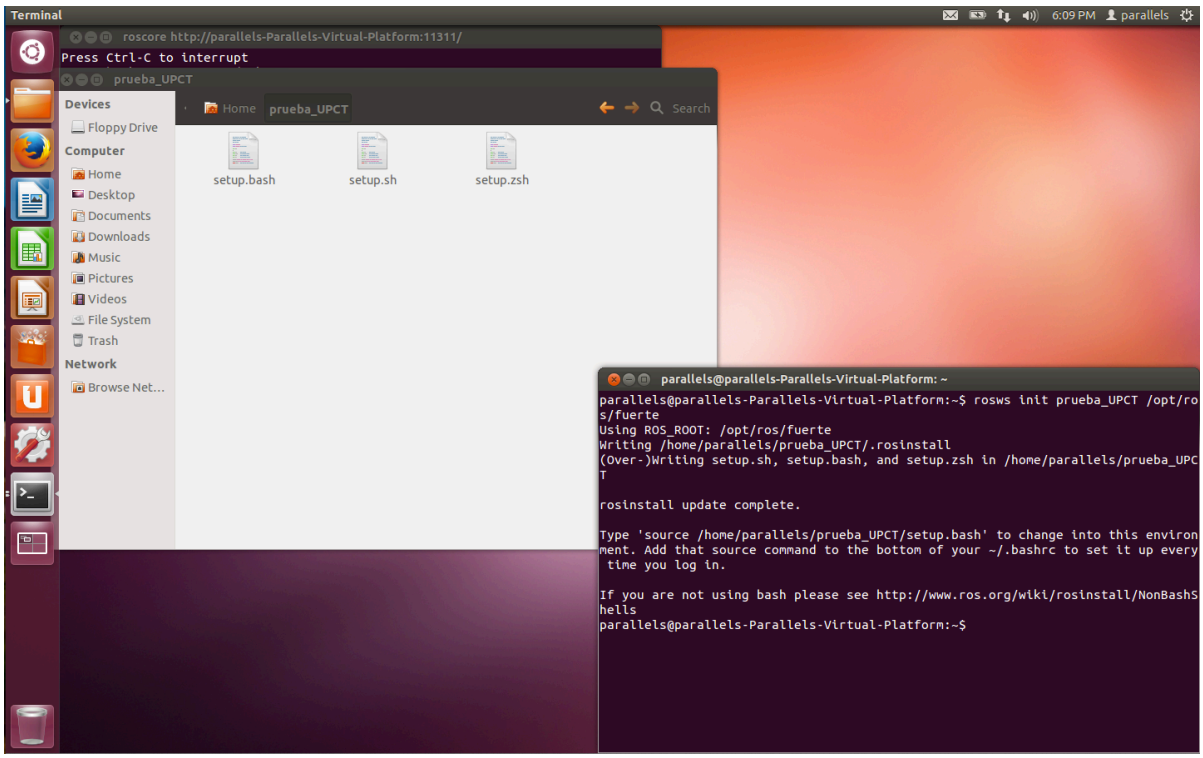
Para crear un nuevo espacio de trabajo utilizaremos el comando *rosws,* que se utiliza de la forma:

$ rosws init [nombre del espacio de trabajo] [localización de los ficheros de ros]

Por ejemplo, para crear un espacio de trabajo llamado prueba_UPCT lo haremos de la forma:

$ rosws init prueba_UPCT /opt/ros/fuerte

Al ejecutar este comando, automaticamente se crearan una serie de archivos necesarios dentro de una carpeta localizada en home/nombre_de_usuario/prueba_UPCT, en este caso nos creará setup.bash, setup.sh, setup.zsh además de un archivo oculto denominado .rosinstall dentro de esta carpeta.



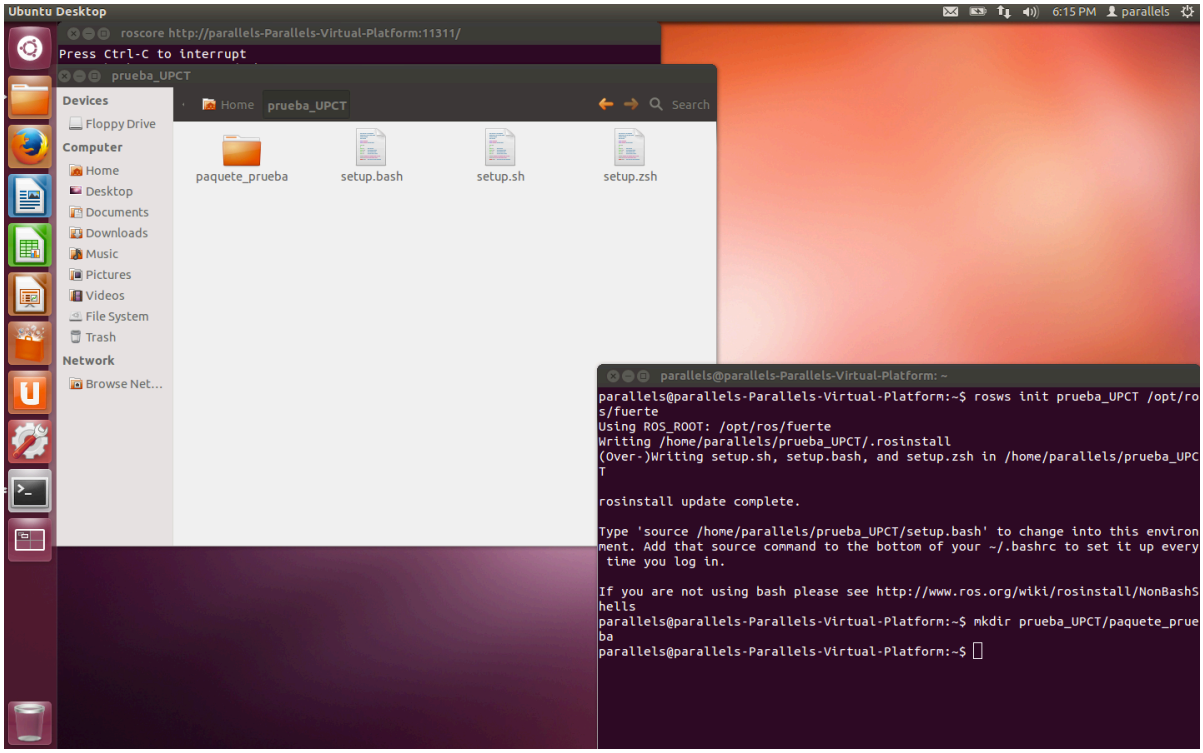
Muestra de como crea la carpeta en nuestro sistema de ficheros.

6.2 Sistema de archivos

Utilizando el comando rosws visto en el punto anterior, todos los paquetes serán incluidos de forma automática dentro de la variable ROS_PACKAGE_PATH cada vez que llamamos al archivo setup.bash del espacio de trabajo necesario.

Por ejemplo, para poder crear una subcarpeta dentro de nuestro espacio de trabajo, utilizaremos la función *mkdir* y añadiremos la carpeta paquete_prueba, y esto se haría de la siguiente forma

$ mkdir prueba_UPCT/paquete_prueba



Muestra de la subcarpeta creada.

A continuación lo que haremos es seleccionar nuestro espacio de trabajo, para que de forma automática todos los paquetes se incluyan en el directorio anterior.

$ rosws set home/Nombre_de_usuario/prueba_UPCT/paquete_prueba

$ source /prueba_UPCT/setup.bash

si queremos también se puede configurar el programa para dejar configurado el entorno de forma automática para cada vez que se abra un nuevo terminal, para ello utilizaremos las siguientes funciones

$ echo"source/opt/ros/fuerte/setup.bash">>~/.bashrc
$ echo"exportROS_PACKAGE_PATH=~/miworkspace:$ROS_PACKAGE_PATH">>~/.bashrc
$ echo"exportROS_WORKSPACE=~/miworkspace">>~/.bashrc
$ echo"exportROS_HOSTNAME=localhost">>~/.bashrc
$ echo"exportROS_MASTER_URI=http://localhost:11311">>~/.bashrc

6.3 Packages y Stacks

Los Packages y los Stacks son los que anteriormente denominábamos paquetes y pilas, pero le hemos dejado su nombre original para una mayor facilidad a la hora de familiarizarnos en esta parte del documento.

Para ir a una ubicación específica dentro de un package o stack usaremos la función *roscd,* que se usa de la forma:

$ roscd Localizacion_del_fichero

Si utilizamos esta función sin argumentos, nos llevará directamente al espacio de trabajo

Para ver una lista con los ficheros contenidos dentro de un package o stack teclearemos el código *rosls*

$ rosls Localizacion_del_fichero

Para esta última función existe una pequeña herramienta ya incorporada que nos ayudará en nuestro trabajo, ya que podemos empezar a escribir el nombre de los paquetes o las pilas y cuando tengamos mas de dos letras escritas, si pulsamos la tecla tabulador, él nos completará el nombre de forma automática.

6.3.1 Rospack

Rospack es una herramienta que sirve para recuperar información en los paquetes de ROS, de forma parecida a como se haría bajo el entorno Linux con los comandos *cd* o *ls*. Podemos encontrar una amplia variedad de comandos que van desde la localización de paquetes de ROS en el sistema de archivos, a un listado de pilas disponibles.

Una función que nos puede ayudar mucho sería:

$ rospack help

Donde nos mostrará todos lo comandos disponibles bajo esta función.

Un ejemplo de uno de los comandos disponibles sería, en el caso de querer encontrar un paquete, y se utilizaría de la forma:

$ rospack find Nombre_del_paquete

6.3.2 Rosstack

Rosstack es una herramienta de línea de comandos que nos permite recuperar información sobre las pilas en ROS. Implementa una amplia variedad de comandos que van desde la localización de las pilas de ROS en el sistema de archivos, a la lista de pilas disponibles para el cálculo del árbol de dependencia de las pilas.También se utiliza en ROS para el cálculo de la información y para construir las pilas.

Su uso es muy parecido al descrito en el apartado anterior con los packages, solo habría que sustituir pack por stack, es decir usar el comando:

$ rosstack

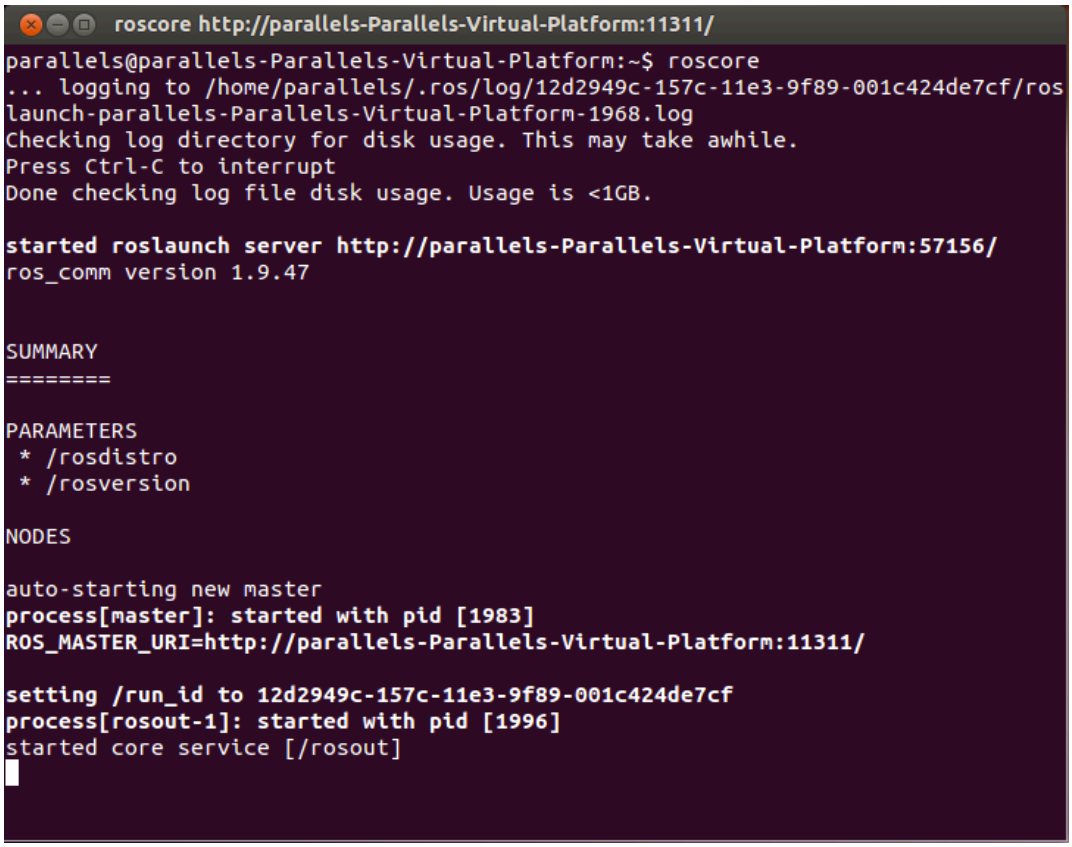
6.4 Motor de ejecución

El motor de ejecución es una colección de nodos y programas básicos necesarios para poder trabajar con ROS, y es esencial para que todos los nodos se puedan comunicar entre si.
Entre los nodos y programas básicos que se inician con el motor de ejecución podemos encontrar el master, el servidor de parámetros y el nodo rosout que es el encargado de trabajar con toda la información del registro de ROS.

Este motor de ejecuta abriendo un terminal exclusivamente para él e introduciendo:

$ roscore

Una vez introducido dicho comando, deberíamos ver algo parecido a esto:



Muestra de roscore ejecutandose.

6.5 Obtener datos

6.5.1 Nodos

Un nodo es como si fuera un ejecutable dentro del paquete de ROS, usa la librería cliente de ROS para comunicarse con otros nodos, y estos pueden publicar o subscribirse a un tópico, además de usar cualquier servicio.
ROS nos permitirá usar nodos creados con otros lenguajes de programación (phyton y c++)
Para poder acceder, modificar y crear cualquier nodo necesitaremos los paquetes: roscode, rosnode y rosrun.

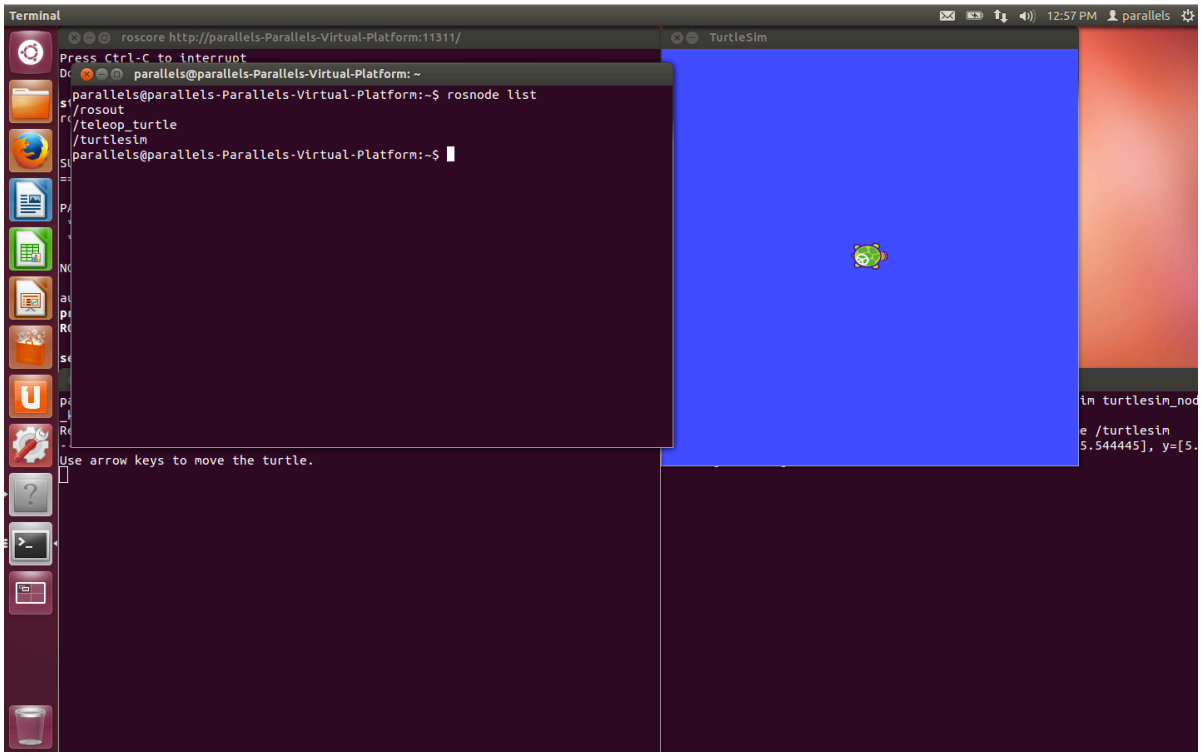
El primer paso será activar *roscore* tal y como hemos visto en el punto anterior.

Justo después activaremos nuestro programa de prueba *turtle* y el nodo *teleop_turtle* para poder ver su funcionamiento.

Ahora vamos a verificar qué nodos están activos, para ello utilizaremos rosnode, introduciendo en el terminal:

$ rosnode list

y se debería ver una respuesta:



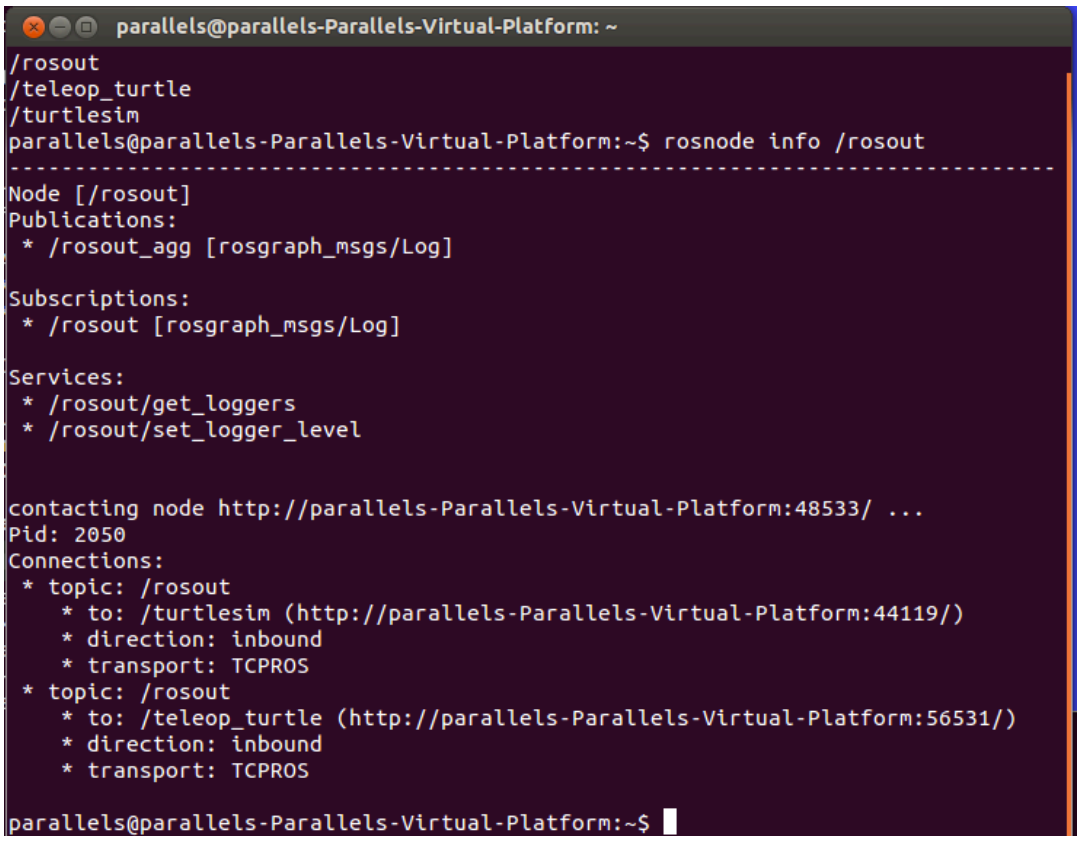
Muestra de los nodos activos.

Esto significa que en este caso está activo el nodo “rosout” (es un nodo que siempre tiene que estar activo, y es el encargado de recoger y registrar todos los nodos de depuración), y los nodos turtlesim (nos muestra la tortuga por pantalla) y teleop_turtle (encargada de mover la tortuga cuando pulsemos las teclas de dirección de nuestro equipo).

Para ver la información de cualquier nodo, podemos usar el comando:

$ rosnode info /rosout

En este ejemplo, lo que se ha pedido es al paquete rosnode que nos proporcione información sobre el nodo rosout, y obtendremos una pantalla parecida a esta



Información del nodo rosout.

como podemos ver en el ejemplo, nos da toda la información relativa a las publicaciones, sus subscripciones, servicios, y la forma de comunicarse con este.

Para activar cualquier nodo, usaremos el comando rosrun, escribiéndolo en el terminal de la forma:

$ rosrun [nombre_paquete] [nombre_nodo]

Por ejemplo si tenemos un nodo llamado sensor_temperatura, y está dentro del paquete turtlesim, para activarlo se haría del modo que sigue:

$ rosrun turtlesim sensor_temperatura

y si en un nuevo terminal introducimos:

$ rosnode list

Veremos como a nuestra lista de nodos se ha añadido un nuevo nodo que tendría por nombre /sensor_temperatura

También podemos cambiar el nombre a un nodo en cualquier momento, el único requisito es que no esté en uso, por ejemplo, vamos a cambiar el nombre de sensor_temperatura a sensor_interno, para ellos usamos el comando:

$ rosrun [nombre_paquete] [nombre_nodo] __name:=[nuevo_nombre]

En nuestro ejemplo nos quedaría:

$ rosrun turtlesim sensor_temperatura __name:=sensor_interno

Por lo que si volvemos a poner en el terminal rosnode list, veremos como el nombre de nuestro sensor se ha cambiado por /sensor_interno

6.5.2 Tópicos

Los tópicos, o también llamados temas, es el sistema usado por los nodos para comunicarse entre ellos.

Lo primero que debemos hacer para poder comprender los tópicos, es activar los nodos que queramos usar, para ello usamos la función del apartado anterior:

$ rosrun [nombre_paquete] [nombre_nodo]

Por facilidad de comprensión y usabilidad, al igual que en los ejemplos anteriores, vamos a hacer uso de los archivos de prueba llamados turtlesim, y turtle_teleop.

Ahora podremos usar las flechas del teclado para describir una trayectoria (en caso que no puedas usar las flechas para dirigirla, prueba a tener activa la pantalla de terminal correspondiente al nodo turtle_teleop_key).

Para poder ver más a fondo su funcionamiento, usaremos el paquete rqt_graph, cuya función es la de crear gráficos dinámicos del sistema sobre lo que va a suceder en tiempo real.

Para asegurarnos de que tenemos este paquete instalado usamos la siguiente función de terminal:

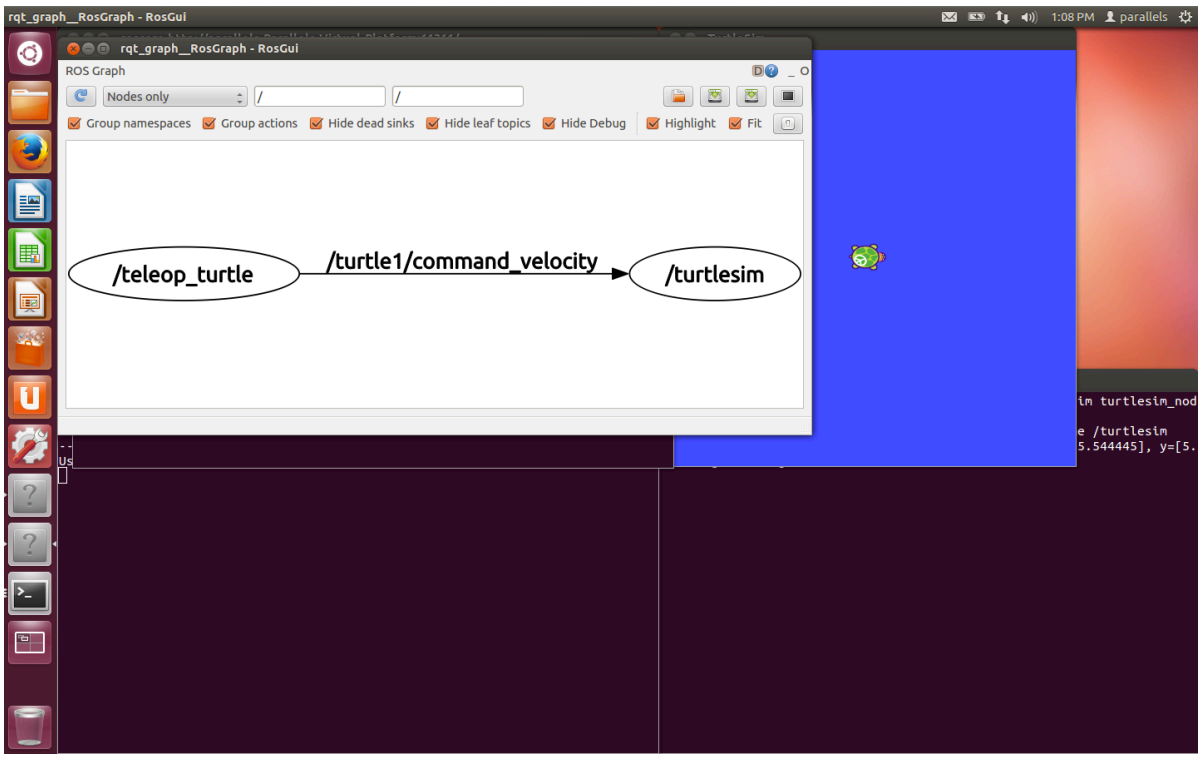
$ sudo apt-get install ros-<distribucion-usada>-rqt

Sustituyendo distribucion-usada, por tu distribución de ROS (Fuerte, Groovy, etc.)

Una vez verificado, o en su caso instalado el paquete, procedemos a usarlo, tecleando en un nuevo terminal:

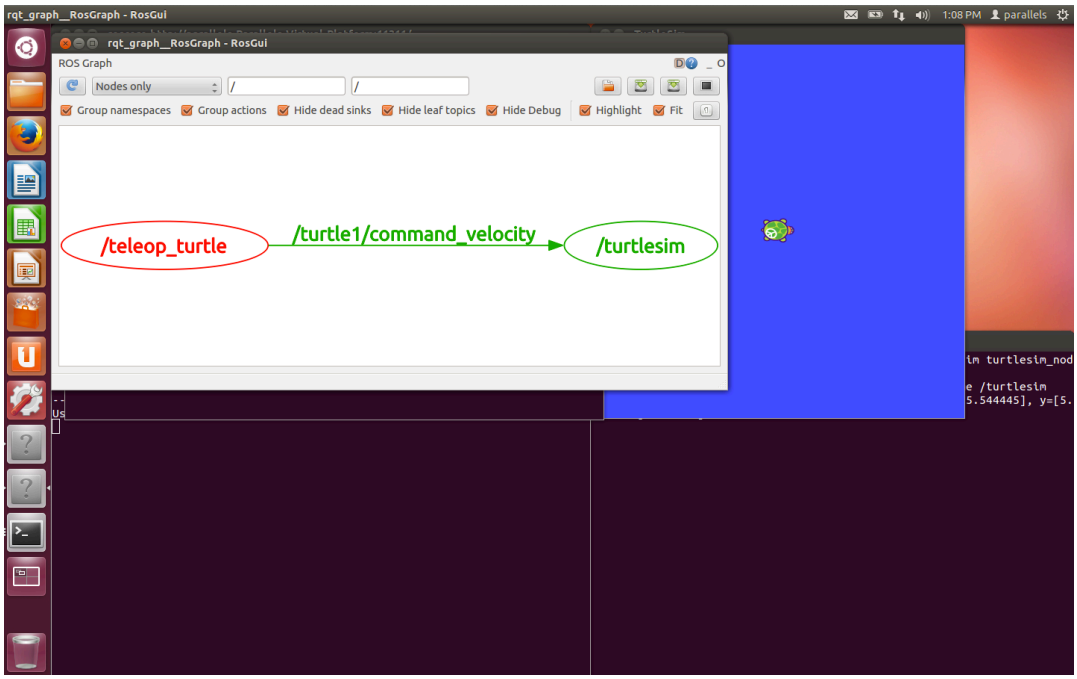
$ rosrun rqt_graph rqt_graph

y el programa debería mostrar una pantalla parecida a la siguiente:



Muestra de rqt_graph con los nodos en uso.

Si pasas el cursor por encima de turtle1/command_velocity/ te mostrará en diversos colores los nodos y los tópicos de los que hace uso. Tal y como se puede observar, los nodos /turtlesim y /teleop_turtle, se comunican mediante el tópico turtle1/ command_velocity/



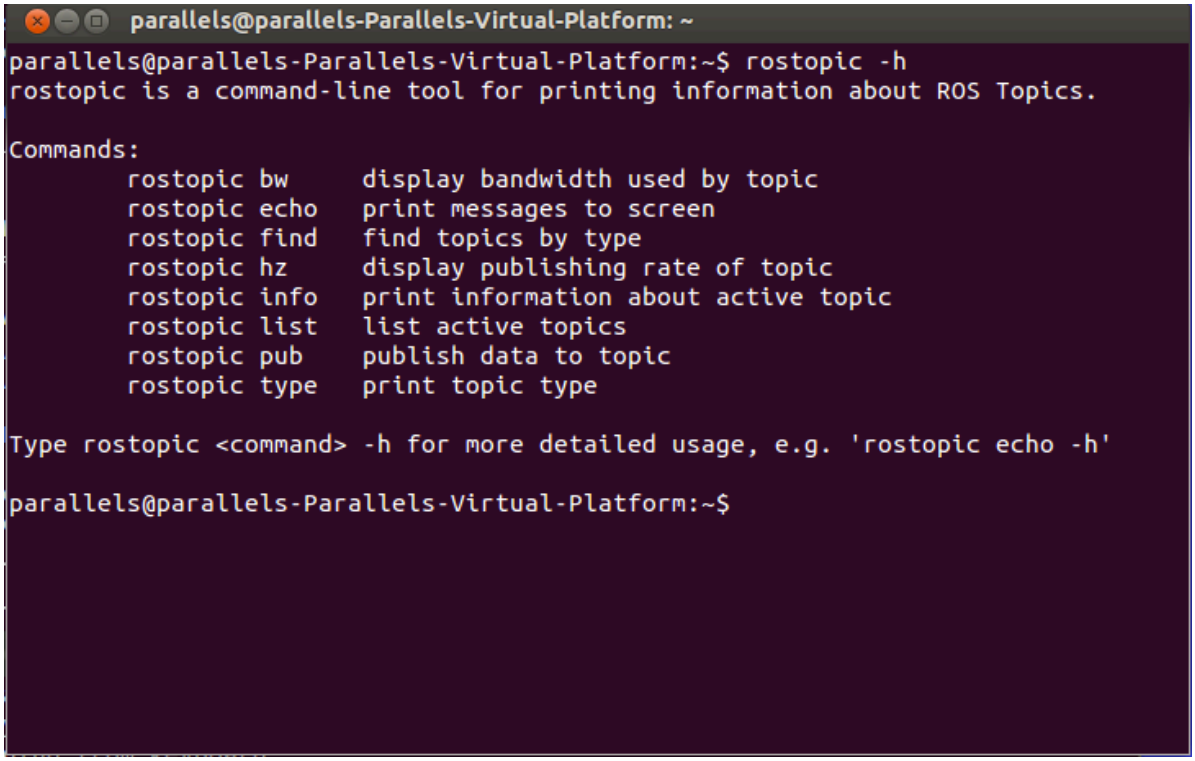
Muestra de los nodos en uso colocando el cursor encima.

Una vez realizada una introducción general sobre los tópicos, procedemos a realizar una visión mas a fondo del comando rostopic.

El comando rostopic nos proporciona información acerca de los tópicos, y podemos acceder a las diferentes opciones que nos ofrece esta función con el comando:

$ rostopic -h

Donde nos mostrará la ayuda de esta función.



Comando de ayuda de rostopic.

Vamos a ver mas en detalle que es lo que hace la mayoría de estos comandos.

- rostopic echo

Esta función nos muestra en pantalla los datos publicados en un tópico.

La función se usa de la forma:

$ rostopic echo [topico]

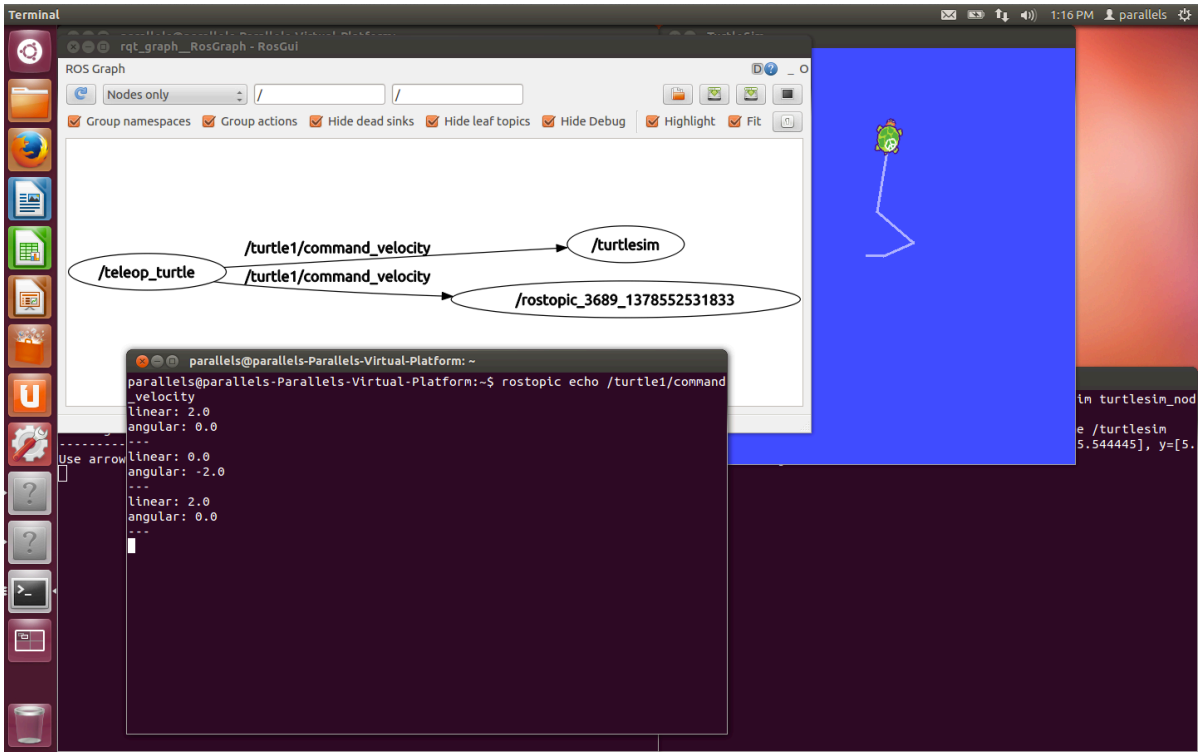
Donde lo único que hay que hacer es sustituir donde pone tópico, por el tema que quieres ver los datos. Por ejemplo, si queremos ver los datos del tópico command_velocity, deberemos introducir la siguiente función

$ rostopic echo /turtle1/command_velocity

Es posible que no aparezca nada en la pantalla, en el caso de que ocurra esto posiblemente es que no hay nada guardado en el tópico, para solucionar este

problema solo debemos ir al terminal donde tenemos activo turtle_teleop_key, y pulsar alguna dirección, por ejemplo la flecha hacia arriba, y ahora veremos como en el terminal de rostopic echo, empiezan a escribir datos.

Si en este momento miramos el terminal de rqt_graph, podremos ver gráficamente como rostopic echo, se ha subscrito al tópico /turtle1/command_velocity.



Muestra de como el nuevo nodo se ha subscrito al principal.

- rostopic list

Este comando nos devuelve una lista de todos los tópicos a los que está subscrito y publicando, este comando también tiene su propia ayuda, para acceder a ella solo tenemos que introducir en un terminal nuevo:

$ rostopic list -h

En este caso vamos a ver un ejemplo del modo verboso del sistema

$ rostopic list -v

Si seguimos con el ejemplo del fichero turtle, nos devolverá una respuesta como esta:

Tópicos a los que está subscrito y en los que publica.

Donde podemos ver claramente el numero de publicaciones y subscriptores, así como su tipo.

- rostopic type

La comunicación entre los diferentes nodos que estén relacionados entre si, se debe hacer mediante el mismo tipo de mensaje.

La función se usa de la forma:

$ rostopic type [topic]

En el caso de nuestro ejemplo con el fichero de prueba turtle quedaría:

$ rostopic type /turtle/comand_velocity

a lo que el programa nos responderá:

turtlesim/velocity

puedes ver los detalles de los mensajes usando rosmsg

$ rosmsg show /turtlesim1/Velocity

Y verás una respuesta del sistema parecida a esta:

float32 linear
float32 angular

-rostopic pub

Este comando es el encargado de publicar datos en un tópico que está publicado en ese momento.

El comando se usa de la forma:

$ rostopic pub [topico] [tipo_de_mensaje] [argumentos]

Un ejemplo de uso de este comando en nuestro ejemplo del archivo turtle sería:

$ rostopic pub /turtle1/command_velocity turtlesim/velocity -- 2.0 1.8

Como se puede ver, el ejemplo anterior indica a el archivo encargado del movimiento en el tópico turtle1/command_velocity, que se mueva con velocidad linear 2 y angular 1,8.

en este caso la tortuga realizará un pequeño movimiento, y enseguida se parará, esto es debido a que el programa requiere un flujo constante de comandos a 1HZ para seguir moviendose, pero esto se puede solucionar añadiendo el comando rostopic pub -r:

$ rostopic pub /turtle1/command_velocity turtlesim/Velocity -r 1 -- 2.0 1.8

esto lo que hace es publicar este comando cada 1Hz.

- rostopic hz

Es el encargado de mandar la información a la frecuencia requerida

El comando tiene la forma:

$ rostopic hz [topic]

Vemos ahora como funciona en nuestro ejemplo:

$rostopic hz /turtle1/pose

y veremos:

```
subscribed to [/turtle1/pose]
average rate: 59.354
        min: 0.005s max: 0.027s std dev: 0.00284s window: 58
average rate: 59.459
        min: 0.005s max: 0.027s std dev: 0.00271s window: 118
average rate: 59.539
      min: 0.004s max: 0.030s std dev: 0.00339s window: 177
average rate: 59.492
        min: 0.004s max: 0.030s std dev: 0.00380s window: 237
average rate: 59.463
```

min: 0.004s max: 0.030s std dev: 0.00380s window: 290

Aquí podemos ver que nuestro ejemplo está funcionando a 60 Hz.

Para decirle al programa que deje de enviarnos información, lo deberemos hacer al igual que para cerrar otros, pulsando CTRL + C.

6.5.3 Mensajes

Un mensaje es una estructura de datos simples que sirven para que los nodos se comuniquen entre si.

Los mensajes soportan las mayoría de estructuras de datos usados en la actualidad (enteros, decimales, booleanos, etc.), así como las estructuras anidadas.

Los mensajes son guardados en un fichero msg en un subdirectorio del paquete, y usan el nombre estándar de ROS, por ejemplo std_msgs/msg/string.msg; como medida de seguridad adicional, los mensajes usan MD5, lo que nos permite saber en todo momento si hay alguna parte del mensaje o del nodo que no esté funcionando correctamente, esto nos permite por ejemplo que cuando un nodo se comunica con otro mediante un mensaje, si este está incompleto, el sistema lo puede detectar al momento y no tener en cuenta este mensaje, realizando una petición de que se vuelva a enviar.

Un mensaje puede incluir un tipo especial de mensaje llamado encabezamiento, que incluye algunos metadatos comunes para este tipo de ficheros (numero de identificación, marcas de tiempo, etc.).

6.5.4 Servicios

Los servicios son los encargados de permitir que los nodos envíen peticiones y reciban respuestas.

Tal y como sucede con los tópicos, aquí disponemos de diversas funciones que nos pueden ayudar a comprender y actuar con nuestro sistema.

El comando para acceder a estas funciones funciona de la forma:

$ rosservice [argumento]

Los diferentes argumentos que existen para esta función son:

- list
  Muestra la información acerca de un servicio activo.
- call
  Llama a un servicio con los argumentos seleccionados, por ejemplo para vaciar un servicio, utilizaremos el argumento clear, lo que hará que se borren todos los datos introducidos con anterioridad.

$ rosservice call clear

- type
  Imprime un servicio.
- find
  Encuentra servicios.

6.5.5 Parámetros

Los parámetros nos permiten guardar y manipular los datos introducidos en ROS, hay que tener en cuenta dentro de estos datos se pueden usar datos de tipo integers (enteros), floats (decimales) , boolean (verdadero o falso), dictionaries (diccionarios) y list (listas).

El uso de esta función es de la forma:

$ rosparam [argumento]

los argumentos existentes para esta función son:

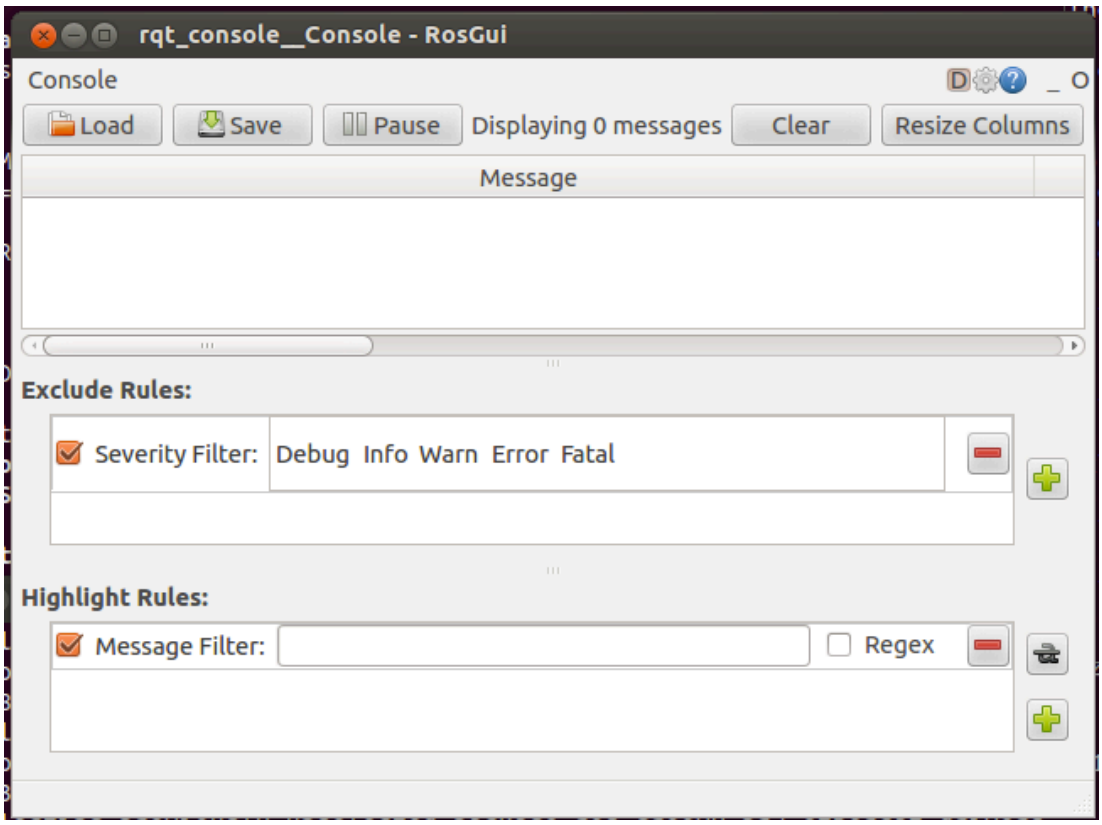
- set
  establece un parámetro.
- get
  obtiene un parámetro.
- load
  carga un parámetro desde un archivo.
- dump
  manda el parámetro a un archivo.
- delete
  borra un parámetro.
- list
  proporciona los nombres de los parámetros en forma de lista.

6.6 Sistema de depuración

Dentro de ROS podemos encontrar un sistema de depuración mediante el cual podemos ver el funcionamiento de cada uno de los nodos y sus interacciones con otros para poder comprobar que el programa se comporte de la forma esperada.

Para activar dicho sistema utilizaremos el código:

$ rqt_console



Sistema de depuración.

6.7 Trabajar con datos

Para trabajar con los datos que nos devuelve el programa mediante sus nodos, utilizaremos la función *rosbag,* lo primero que vamos a ver es como guardar estos datos para poder procesarlos en el momento que queramos.

$ mkdir bagfiles // esto nos genera un directorio denominado bagfiles en nuestro sistema

$ cd bagfiles //accedemos a la carpeta bagfiles

$ rosbag record -a //comenzamos la grabación de datos

Podemos probarlo con nuestro programa de prueba de la tortuga que hemos utilizado en el apartado de primeros pasos con ROS, para ello primero inicializamos el programa, y probamos a mover la tortuga, tras varios movimientos podemos cerrar la aplicación rosbag y ver el archivo generado (el fichero lo podemos encontrar dentro de nuestra carpeta /home/bagfiles, y como nombre tendrá la fecha y hora de creación del fichero).

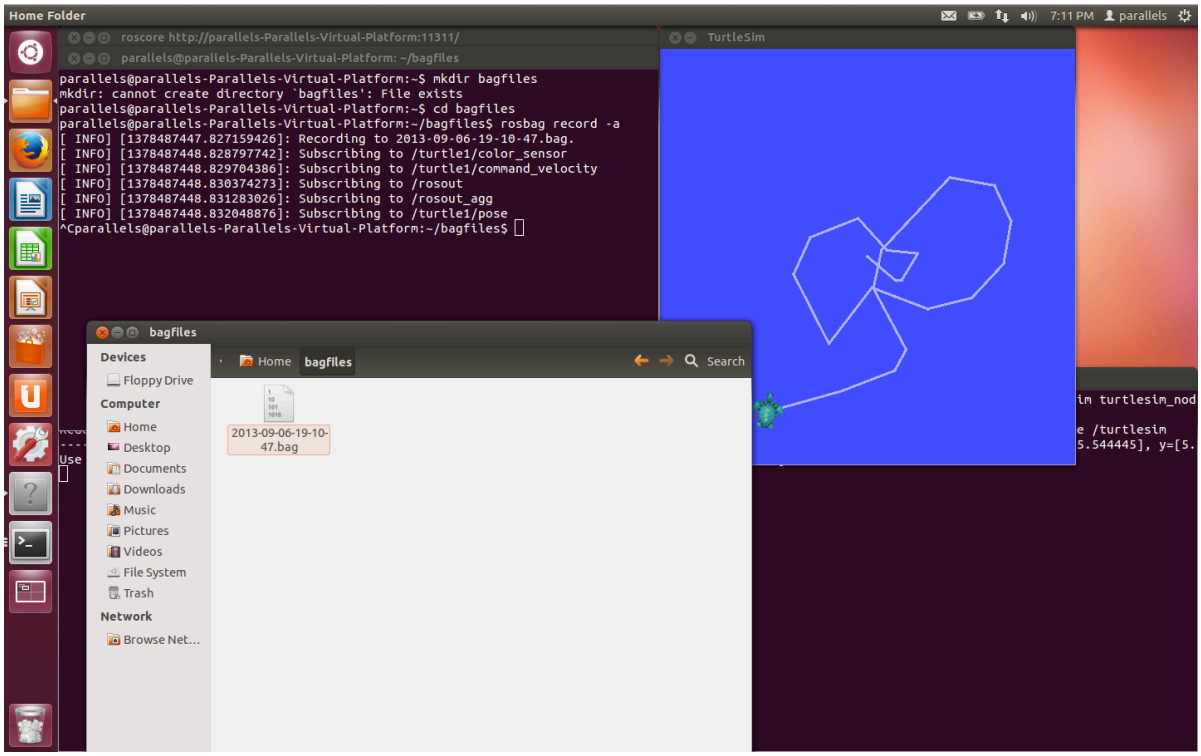


Imagen del fichero generado.

Para visualizar los datos guardados, solo tendremos que usar la función:

$ rosbag info Nombre_del_fichero

En nuestro caso quedaría:

$ rosbag info 2013-09-06-19-10-47.bag

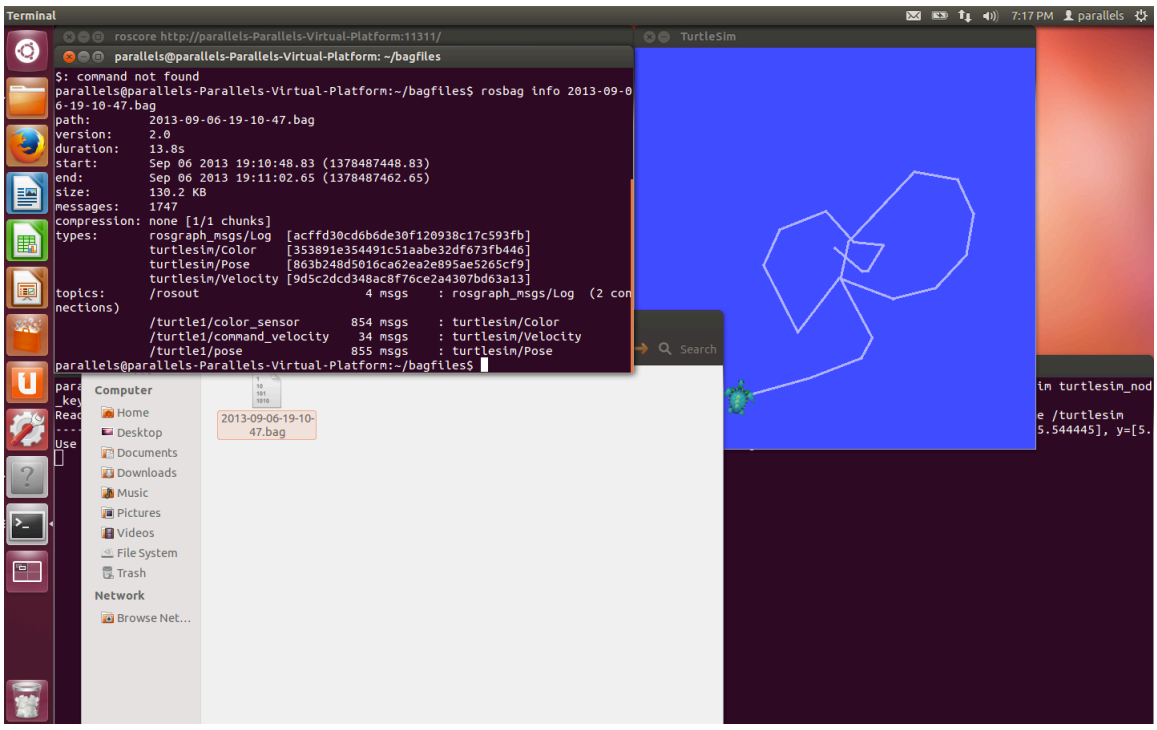


Imagen de la información extraída del fichero generado.

Para volver a recuperar los datos guardados, para que el programa siga por donde se quedó solo tenemos que añadir play tras el comando rosbag

$ rosbag play Nombre_del_fichero

Que en nuestro ejemplo sería asi:

$ rosbag play 2013-09-06-19-10-47.bag

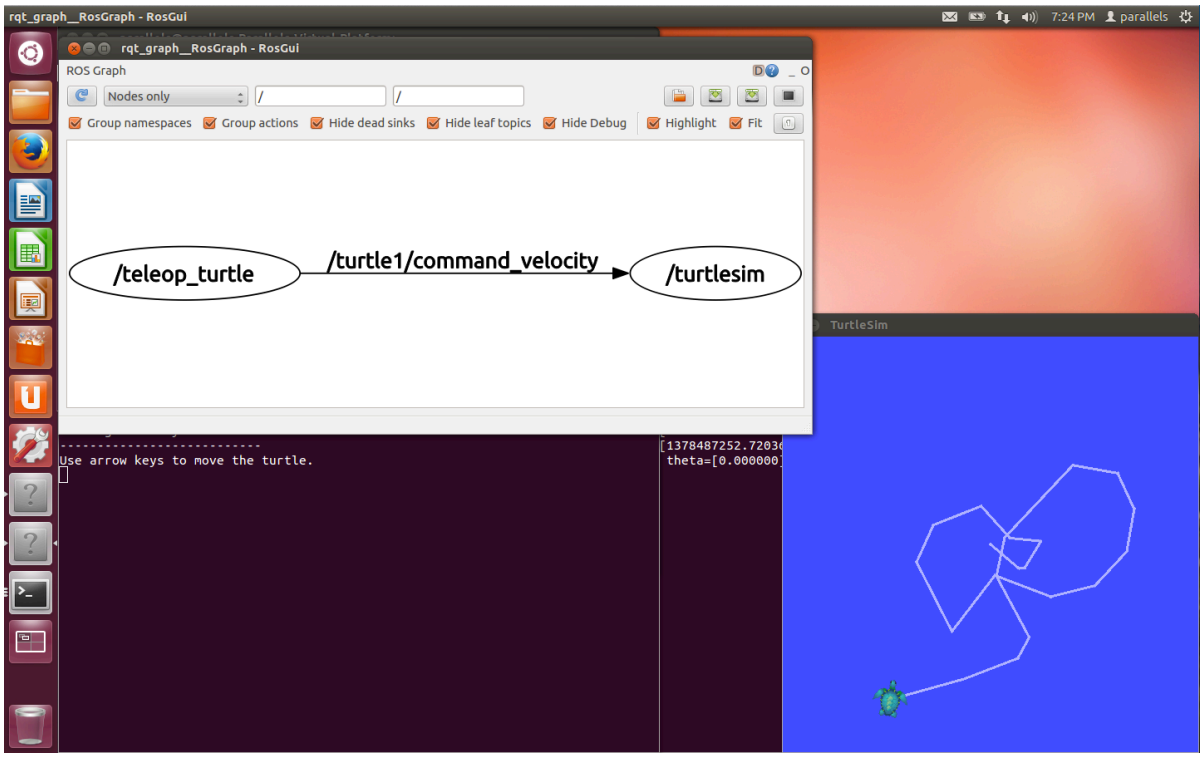
6.8 Visualizando los datos

ROS tiene una función que nos puede ayudar mucho a la hora de saber como trabaja nuestro sistema, y de si lo está haciendo de forma correcta, esta función es rqt_graph, y nos mostrará de forma gráfica los nodos que están en funcionamiento, sus dependencias y su forma de trabajar.

Para utlizar esta función solo tenemos que introducir en un terminal nuevo la función

$ rqt_graph

En la imagen que podemos ver a continuación podemos ver un ejemplo de su funcionamiento con el programa turtle



Representación gráfica de los nodos activos.

Podemos ver de forma gráfica como el nodo /teleop_turtle se encarga de enviar la información mediante mensajes al nodo /turtlesim

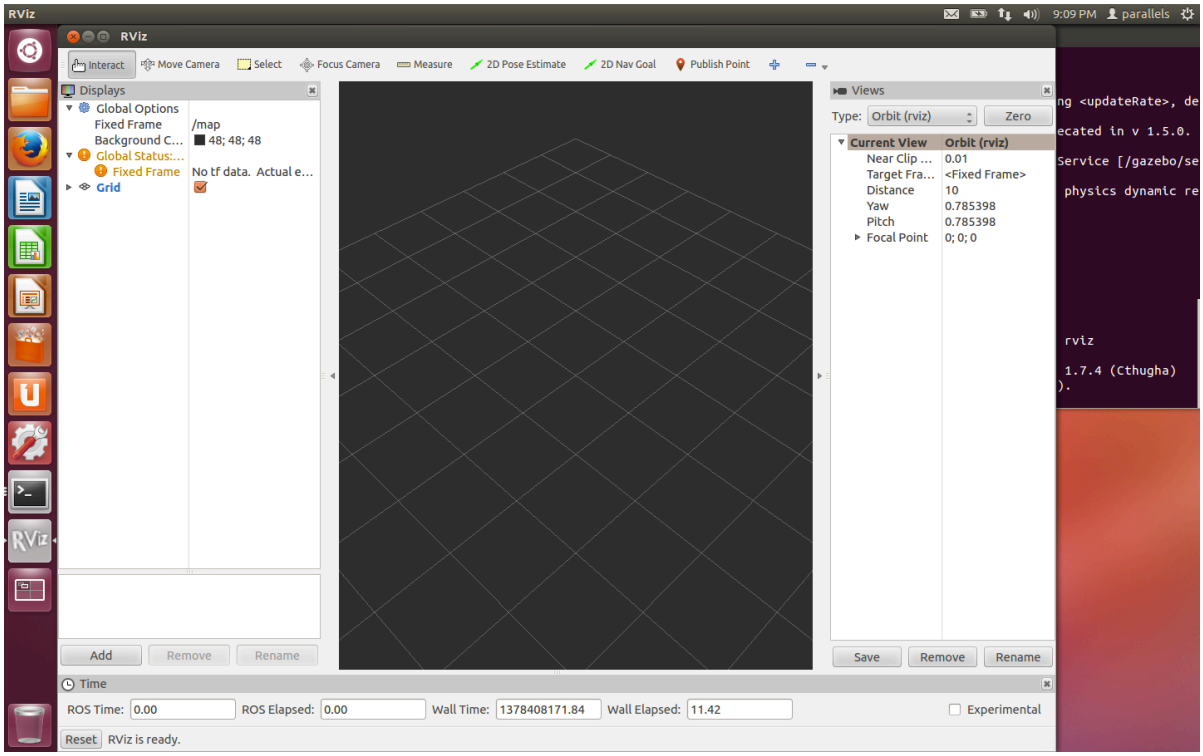
7. Virtualizando ROS

7.1 Rviz

ROS dispone de una herramienta de visualización en 3D llamada RVIZ que posibilita que nuestro robot Qbo, o prácticamente cualquier otra plataforma robótica, pueda ser representada en imagen 3D, respondiendo en tiempo real a lo que le ocurre en el mundo real.

RVIZ se puede usar para mostrar lecturas de sensores, datos devueltos por la visión estereoscópica (Cloud Point), hacer SLAM (localización y mapeo simultáneo) evitando obstáculos, etc. Esta herramienta dispone así mismo, de muchísimas opciones de configuración.

Para generar cada modelo de robot en particular, se tiene que editar en un archivo XML y escribir en URDF (Unified Robot Description Format) donde se especifican las dimensiones del robot, los movimientos de las articulaciones, parámetros físicos como masa e inercia, etc. En el caso de robots con muchas articulaciones ROS dispone de otra herramienta llamada TF que es una biblioteca que facilita la elaboración en estos casos. De todas formas ROS tiene modelos ya creados para probar con RVIZ.



Lo primero que tenemos que hacer es inicializar ROS, abriendo una ventana desde la terminal y escribiendo en ella:

$ roscore

Luego abrimos otra ventana desde terminal y escribimos:

$ rosrun rviz rviz

Cuando terminemos de trabajar con el programa, para cerrarlo simplemente tendremos que ir a la ventana de terminal donde lo activamos y pulsar la combinación de teclas Ctrl + C.

7.2 Gazebo

Gazebo[7] es un simulador gráfico de código abierto que se integra perfectamente con ROS y nos permite virtualizar nuestro robot, así como el entorno sobre el que va a funcionar, permitiéndonos probar el funcionamiento de este en un entorno controlado antes de llevarlo a la realidad, ayudándonos a ahorrar dinero y esfuerzos a la hora de crear un sistema desde cero o reutilizar un sistema ya generado con anterioridad.

Primero procedemos a instalarlo en nuestro sistema.

$ sudo apt-get install ros-groovy-simulator-gazebo

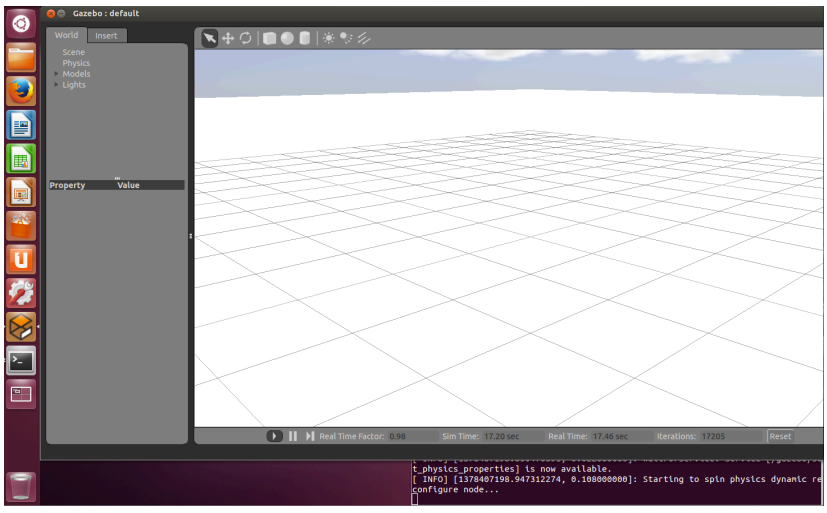
Y ahora procedemos a instalar el robot de prueba

$ sudo apt-get install ros-groovy-pr2-simulator

Tenemos multitud de opciones a la hora de utilizar Gazebo, en este caso vamos a inicializar el programa con un mundo virtual vacío sobre el que pondremos un robot de ejemplo creado por los programadores de ROS (Willow Garage), para ello abrimos una ventana de terminal y ponemos la función:

$ roslaunch gazebo_worlds empty_world.launch

Esto nos abrirá una ventana parecida a esta.



Ventana de Gazebo con un mundo vacío.

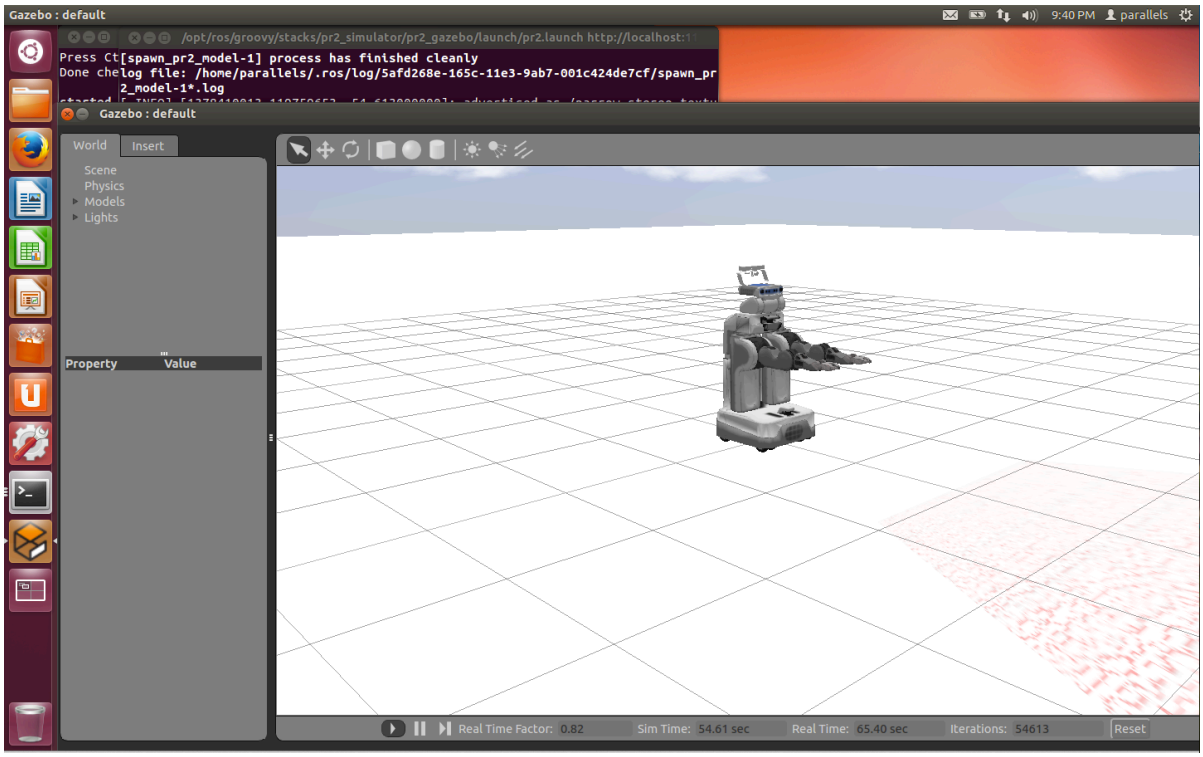
[7] http://www.gazebosim.org

Y ahora vamos a añadir el robot citado anteriormente, en este caso el robot PR2

$ rosmake pr2_gazebo

$ roslaunch pr2_gazebo pr2.launch

Una vez que tengamos el robot en nuestro mundo virtual, ya podremos utilizarlo como si estuviéramos en la vida real, o bien mediante nodos automatizados, o bien añadiendo nodos que nos permitan moverlo de forma manual.



Cuando terminemos de trabajar con el programa, para cerrarlo simplemente tendremos que ir a la ventana de terminal donde lo activamos y pulsar la combinación de teclas Ctrl + C.

8. ROS y la docencia

8.1. Aplicaciones docentes

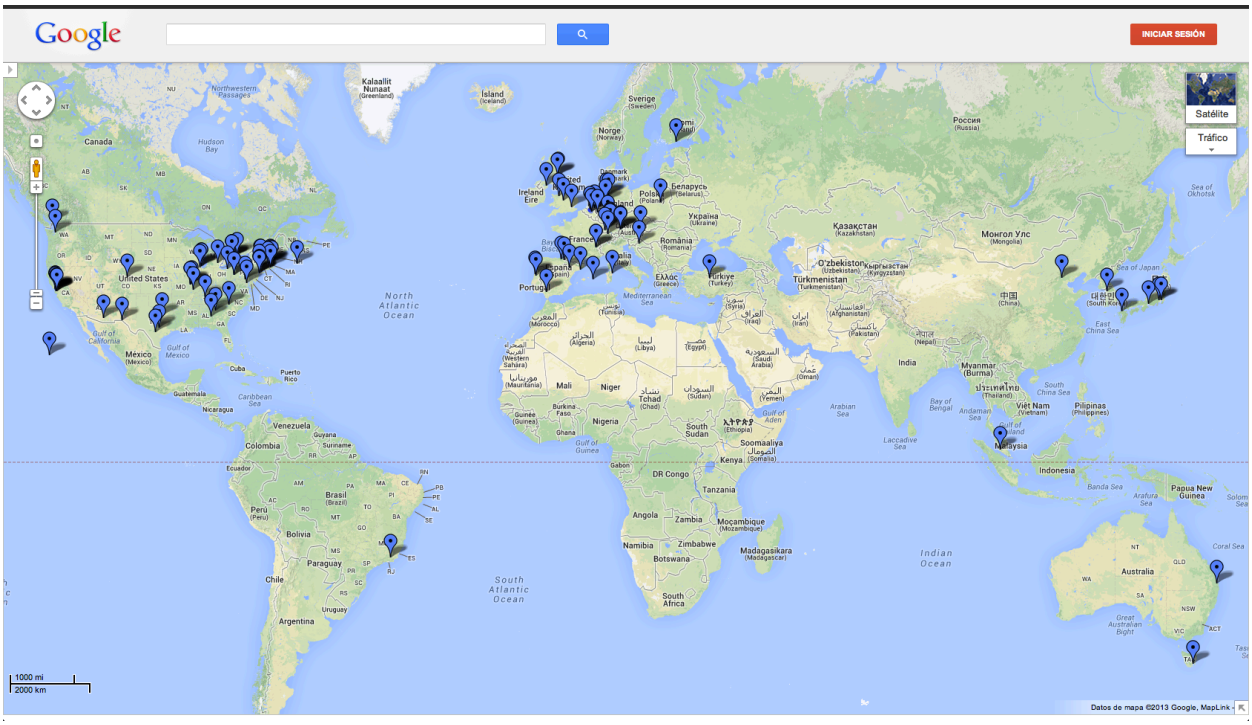
El sistema ROS es un sistema que permite ir desde la base mas simple de la robótica hasta el apartado mas complejo, ya que nos permite utilizar la mayoría de robots que se usan en la actualidad en los centros de enseñanza e investigación, y también nos permite el crear nuestro propio robot desde cero.

En el mes de Septiembre de 2013, hay mas de 65 instituciones docentes de todo el mundo trabajando e investigando con el sistema operativo de ROS,.

En España podemos encontrar universidades como Sevilla, Zaragoza, y Barcelona.

A nivel global encontramos una gran variedad de instituciones docentes como pueden ser: Stanford, Instituto Tecnológico de Massachusetts (MIT) y la universidad de estudios aeroespaciales de Toronto entre otras.

A continuación podemos ver un mapa que se encuentra en la página de ROS, donde se incluye los lugares de donde se han publicado repositorios para el programa.



Mapa de repositorios[8] de ROS

8.2. Sensores soportados

Ros soporta una gran cantidad de sensores de diferentes marcas y funcionalidades, y todos ellos tienen un apartado específico en la pagina web del sistema, y en el caso de que el sensor que tu necesitas usar no está soportado, tienes la opción de generar tu la documentación necesaria para su uso, o bien pedir ayuda en la pagina de soporte de ROS, donde la comunidad seguro que te puede guiar o incluso realizar la documentación necesaria.

Sensores de 1 dimensión
- Sharp IR range finder (via ArbotiX RoboController)

Sensores 2D
- Hokuyo Scanning range finder

[8] https://maps.google.com/maps/ms?ie=UTF&msa=0&msid=209668390659853657363.00049c608b78bc7779683

- SICK LMS2xx lasers
- SICK LMS1xx lasers
- Sick(R) S300 Professional
- Leuze rotoScan laser rangefinder driver (ROD-4, RS4)
- Neato XV-11 Laser Driver
- Sick LD-MRS Laser Driver
- SICK TiM3xx lasers

Sensores 3D
- Mesa Imaging SwissRanger devices (3000/4000)
- OpenNI driver for Kinect and PrimeSense 3D sensors
- Velodyne HDL-64E 3D LIDAR
- Forecast 3D Laser with SICK LIDAR
- PMD Camcube 3.0

Reconocimiento de audio y habla
- hark
- pocketsphinx

Cámaras
- canon_gphoto
- cmucam_png
- dynamic_uvc_cam
- gencam_cu
- GeViCAM stereo camera
- gstreamer camera driver
- gPhoto Driver
- IEEE 1394 Digital Camera
- MatrixVision BlueFOX
- PointGrey Cameras (with FlyCapture2)
- Prosilica Camera
- usb_cam
- uvc_camera
- videre_stereo_cam
- WGE100 camera

Sensores ambientales
- Gill Instruments Windsonic ultrasonic wind sensor
- ce_environment

Sensores de fuerza / par / táctiles
- Schunk LWA 3 Force Torque Controller based on ATI Mini 45
- Nano17 6-axis force/torque sensors
- skin_driver
- Interface to ATI NetFT sensor adapter
- ATI nano 25 and AMTI HE6x6 force plate

Sensores de captura de movimiento
- OptiTrack Motion Capture system using NatNet and VRPN.
- Phase Space optical motion capture system
- VICON motion capture system
- Motion Analysis motion capture system

Sensores de estimación de la posición (GPS/IMU)
- Applanix Position and Orientation System for Land Vehicles
- Bosch Sensortec BMA180 3-axis accelerometer
- Bosch SMI530/540 3-axis sensor
- CH Robotics UM6 IMU
- gpsd_client
- microstrain_3dmgx2_imu
- Xsens MTi node
- Xsens MTI Measurement Unit
- Xsens MTx/MTi/MTi-G devices
- Razor's IMU 9 DOF (Degree of Freedom) board

Fuentes de alimentación
- Carnetix CNX-P2140 DC-DC power supply
- Mini-Box M4-ATX power supply
- Ocean Server Technology Intelligent Battery and Power System

RFID
- UHF RFID Reader

Interfaces de sensores
- Arduino Sensor Interface Module
- ArbotiX RoboController
- Lego NXT Sensors
- Phidgets sensor interface
- Phidgets sensor interface with differential drive
- Phidgets Interface
- libphidgets21
- pmad
- roboard_sensors
- rosserial_arduino
- serializer
- Sensoray 626 analog and digital I/O
- Shadow RoNeX

9. Aplicación

9.1 Creando nuestra aplicación

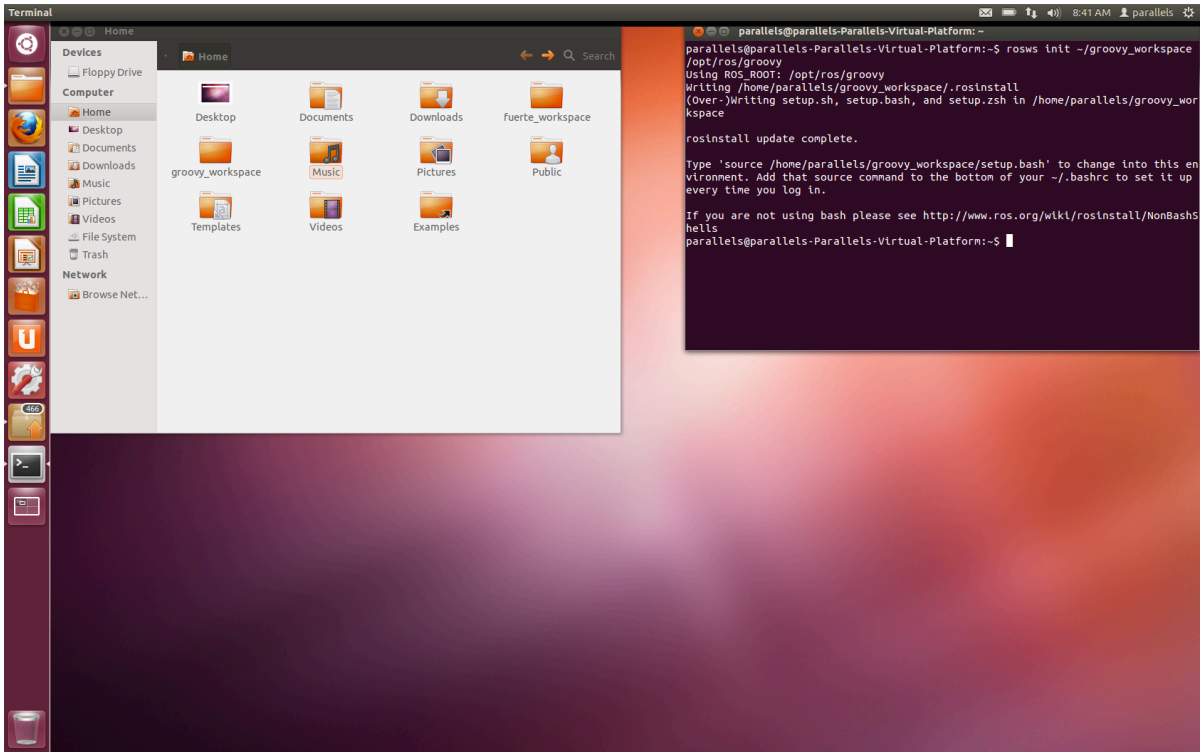
Ahora que ya conocemos el sistema de ROS en profundidad vamos a proceder a crear nuestro propio programa, y que mejor para mostrar su funcionamiento que crear el primer programa típico en todos los sistemas, el denominado “Hola Mundo”, para ello vamos a generar nuestro espacio de trabajo. el paquete que contendrá nuestro programa y un nodo que muestre por pantalla esta frase.

9.2 Preparando el sistema

Lo primero que vamos a hacer es crear nuestro espacio de trabajo, esto nos permitirá tener un lugar donde guardar nuestro programa y poder trabajar tanto con el que vamos a crear ahora como con proyectos personales futuros.

Vamos a crear nuestro espacio de trabajo, indicándole a nuestro sistema que genere todo lo necesario para utilizarlo como un paquete de ROS, al igual que lo tiene en el directorio opt/ROS

rosws init ~/groovy_workspace /opt/ros/groovy



Primeros pasos para generar nuestro espacio de trabajo.

Para crear nuestro directorio utilizaremos el comando *mkdir*, de la forma:

$ mkdir [espacio/de/trabajo]

En nuestro ejemplo hemos decidido crear nuestro espacio de trabajo dentro de una carpeta llamada UPCT en el directorio groovy_workspace.

$ mkdir ~/groovy_workspace/UPCT

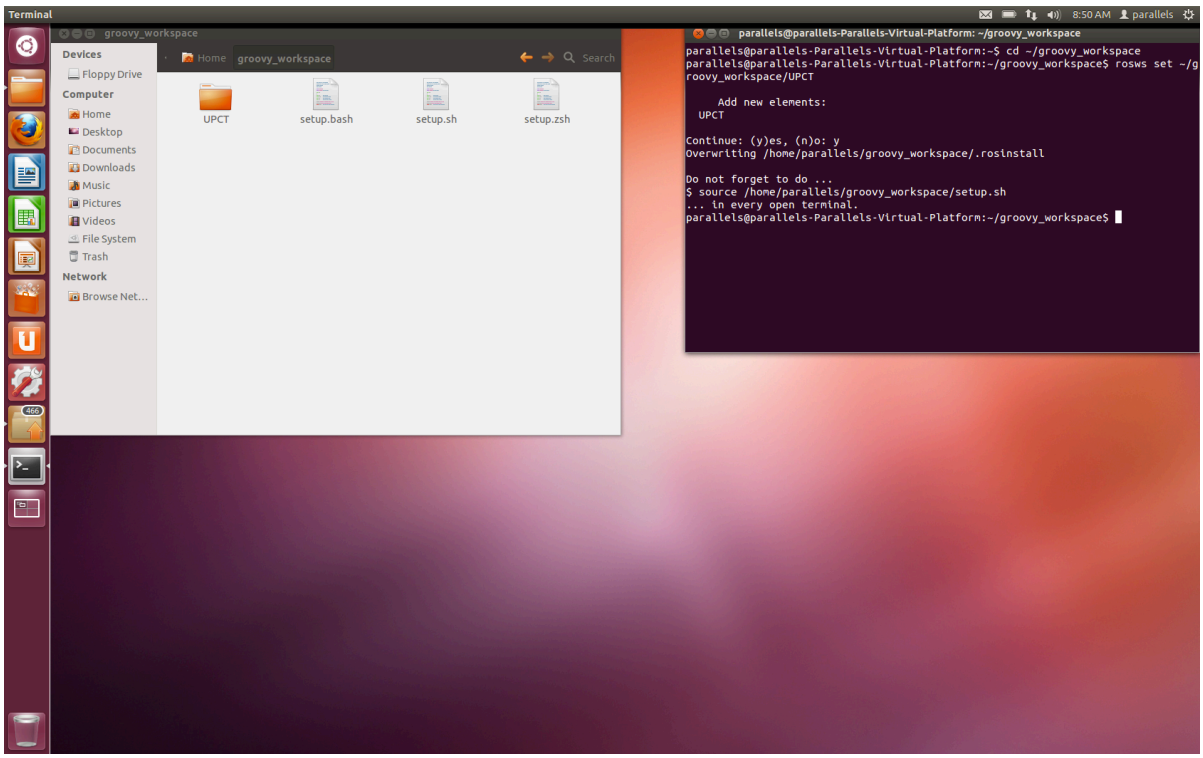


Imagen de la carpeta generada.

Y ahora procedemos a indicarle a nuestro sistema que este directorio recién creado va a ser un espacio de trabajo de ROS.

$ rosws set ~/groovy_workspace/UPCT

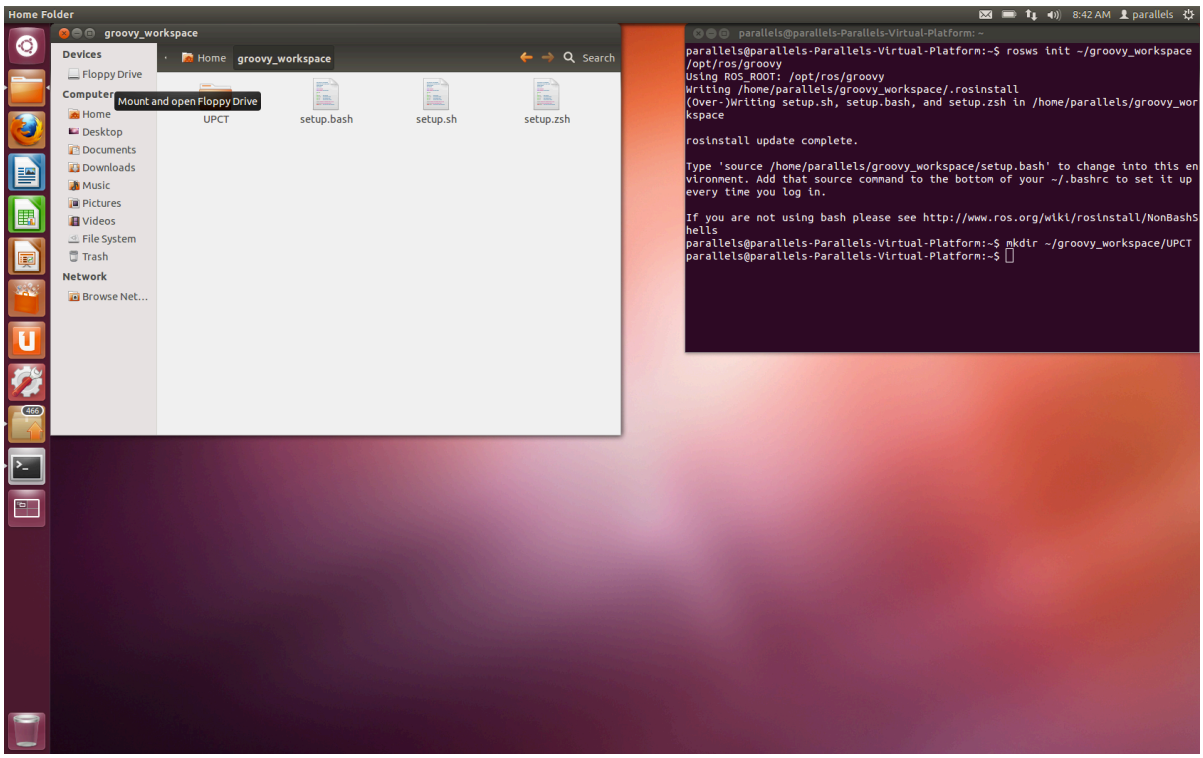


imagen del código introducido

Y ahora lo que hay que hacer es configurar el archivo .bash para indicarle a cada terminal donde debe trabajar.

$ source ~/groovy_workspace/setup.bash

Hay que destacar que deberemos de utilizar este ultimo comando en cada uno de los terminales abiertos que queramos trabajar con nuestro sistema, o bien configurarlo dentro del repositorio general para que lo seleccione de forma automática.

Para confirmar que nuestro directorio lo reconoce ROS como un directorio válido de trabajo utilizaremos la función:

$ echo $ROS_PACKAGE_PATH

A lo que veremos una pantalla parecida a esta:

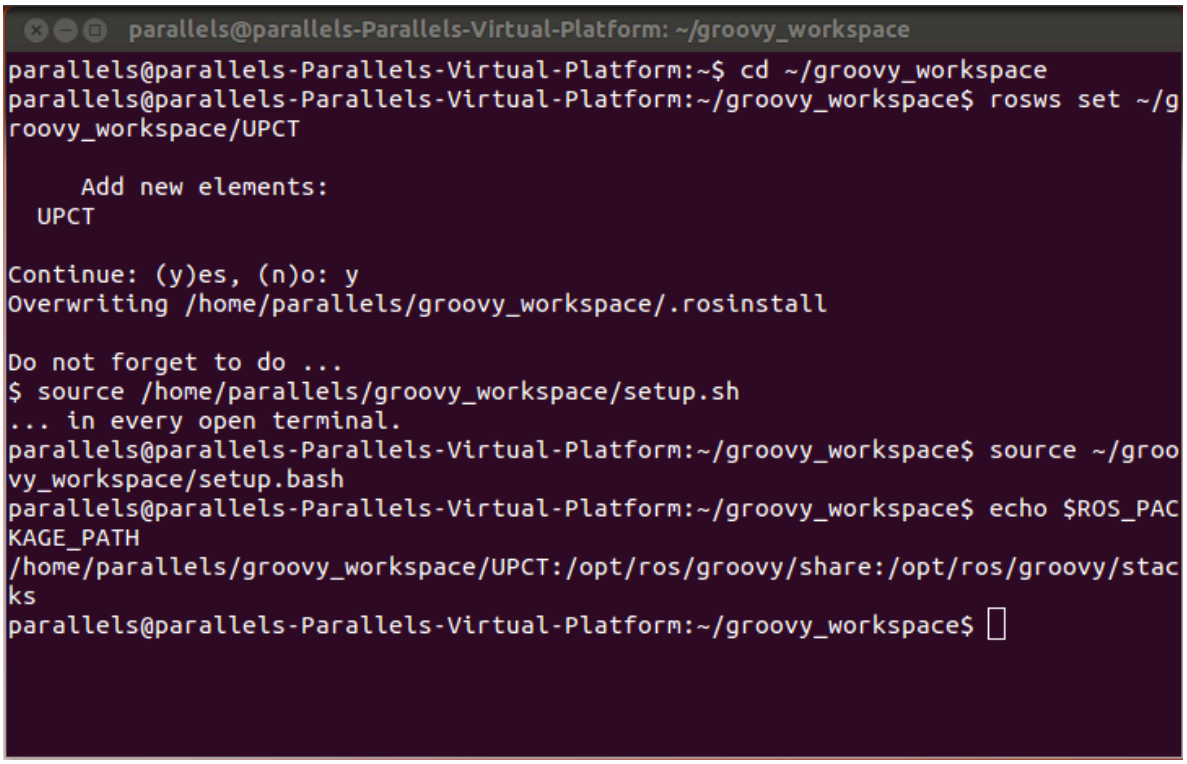


Imagen confirmando que detecta el espacio de trabajo recién generado.

9.3 Creando nuestro paquete

Ahora vamos a crear nuestro paquete, que será capaz de contener nuestro programa.

Lo primero que vamos a hacer es acceder al espacio que hemos creado, para ellos usamos en nuestro caso la función:

$ cd ~/groovy_workspace/UPCT

Y ahora procedemos a crear el paquete que contendrá nuestro futuro programa.

$ roscreate-pkg UPCT_PFC std_msgs rospy roscpp

Mediante esta función generamos un paquete llamado UPCT_PFC que depende de std_msgs, rospy y roscpp.

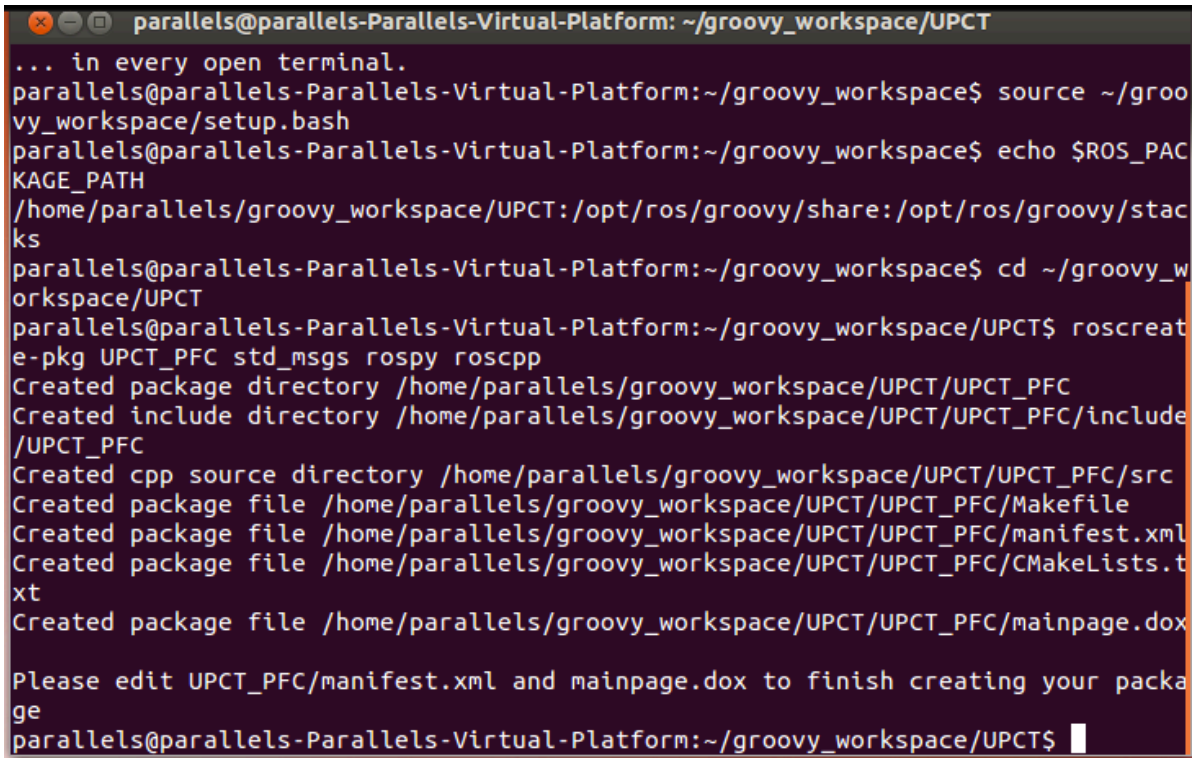


Imagen del sistema generando el paquete.

Ahora lo que vamos a hacer es comprobar que ROS es capaz de encontrar este nuevo paquete

$ rospack find UPCT_PFC

A lo que veremos en pantalla una respuesta como esta:

/home/parallels/groovy_workspace/UPCT/UPCT_PFC

Ahora procedemos a construir nuestro paquete y para ellos solo tenemos que introducir en nuestro terminal

$ rosmake UPCT_PFC

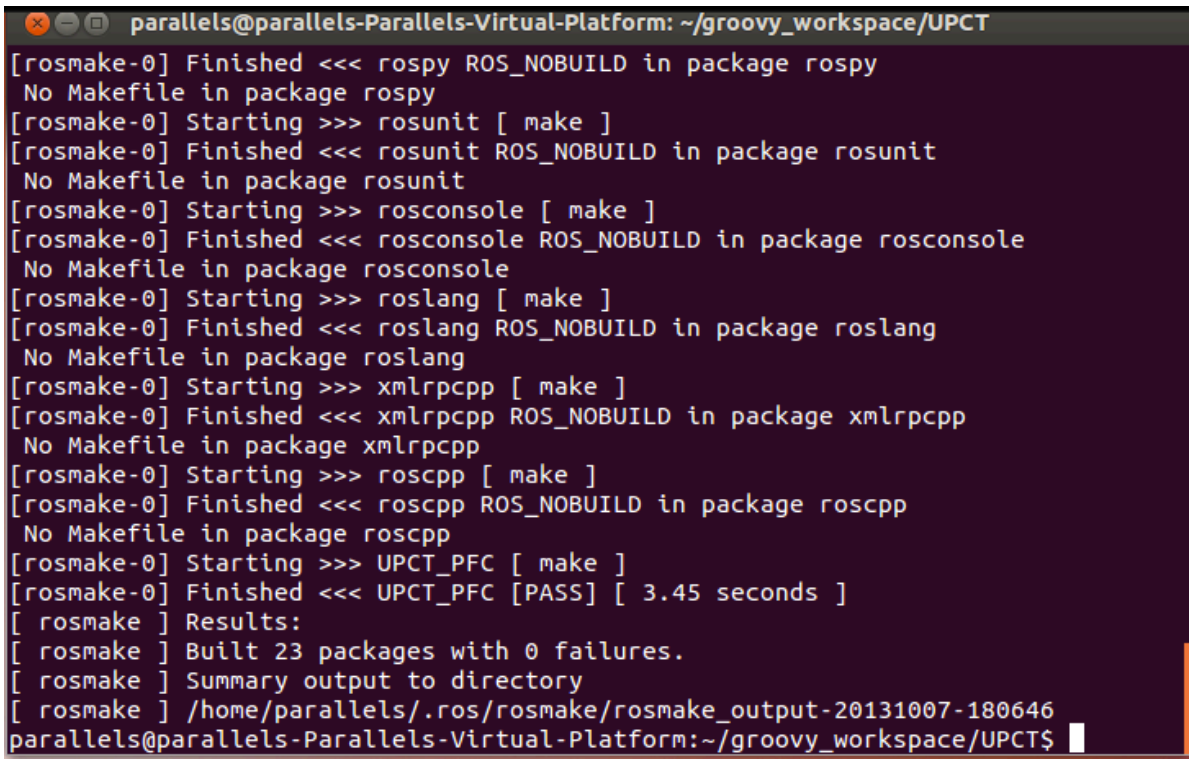


Imagen de ROS creando el paquete.

9.4 Preparando nuestro programa

Ahora vamos a hacer unos pasos opcionales pero altamente recomendables antes de empezar a crear nuestro programa, que es modificar los ficheros *Manifest.xml* y *Mainpage.dox*

En este caso Linux trae su propia función para editar estos documentos llamado gedit

Primero accedemos a la carpeta contenedora

$ cd UPCT_PFC

y ahora usamos la función descrita antes para modificar nuestro fichero

$ gedit manifest.xml

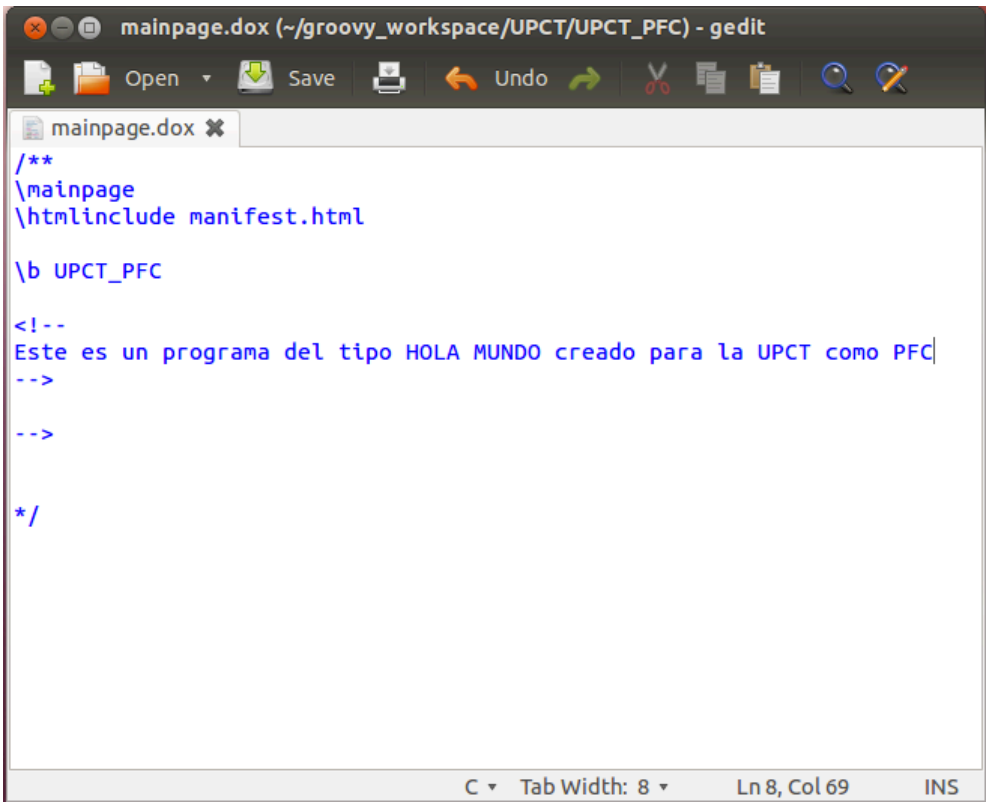


Imagen del contenido de mainpage.dox modificado.

$ gedit mainpage.dox

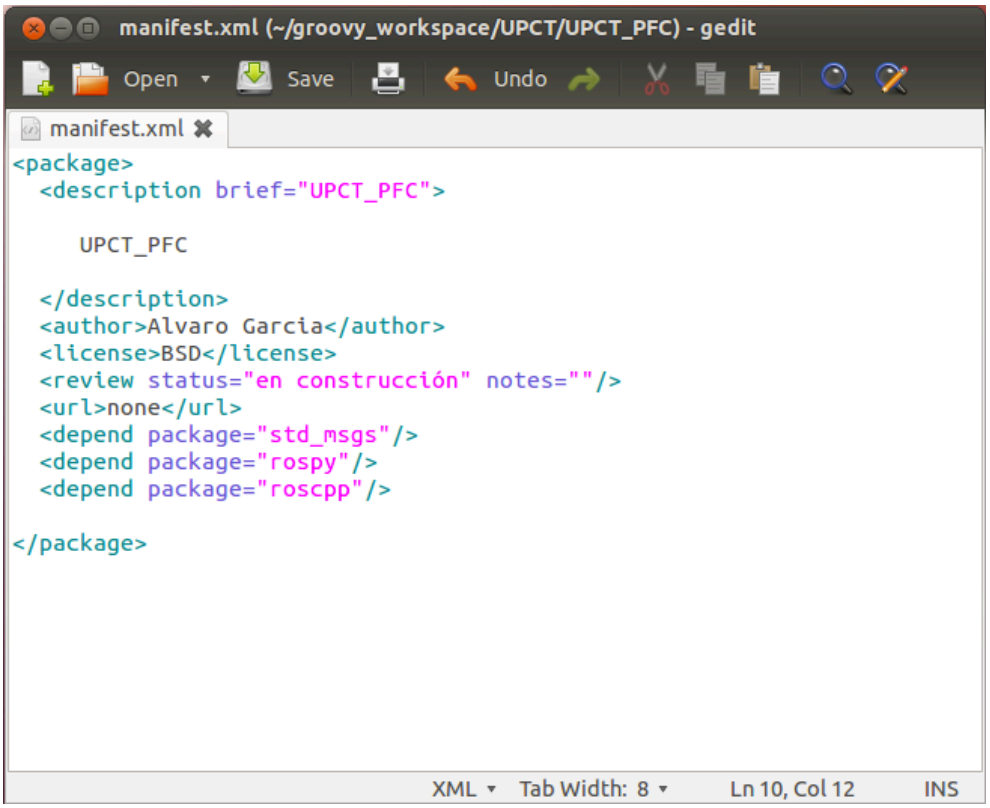


Imagen del contenido de manifest.xml modificado.

Estos ficheros nos ayudan a tener una idea general del proyecto, de su autor y de su forma de funcionar, y puede ayudar a cualquier persona a saber si lo que busca lo va a encontrar en este paquete o no, así como mantener una forma de jerarquizar los paquetes iguales para todo el mundo.

9.5 Construyendo nuestro programa

Ahora que ya tenemos todo listo procedemos a crear nuestro fichero ejecutable, para ello vamos a nuestro sistema de ficheros a /UPCT_PFC / src, y le damos al botón derecho y pulsamos en crear un nuevo documento, documento en blanco.

El siguiente paso es poner el nombre a nuestro fichero en este caso lo llamaremos holamundo.cpp

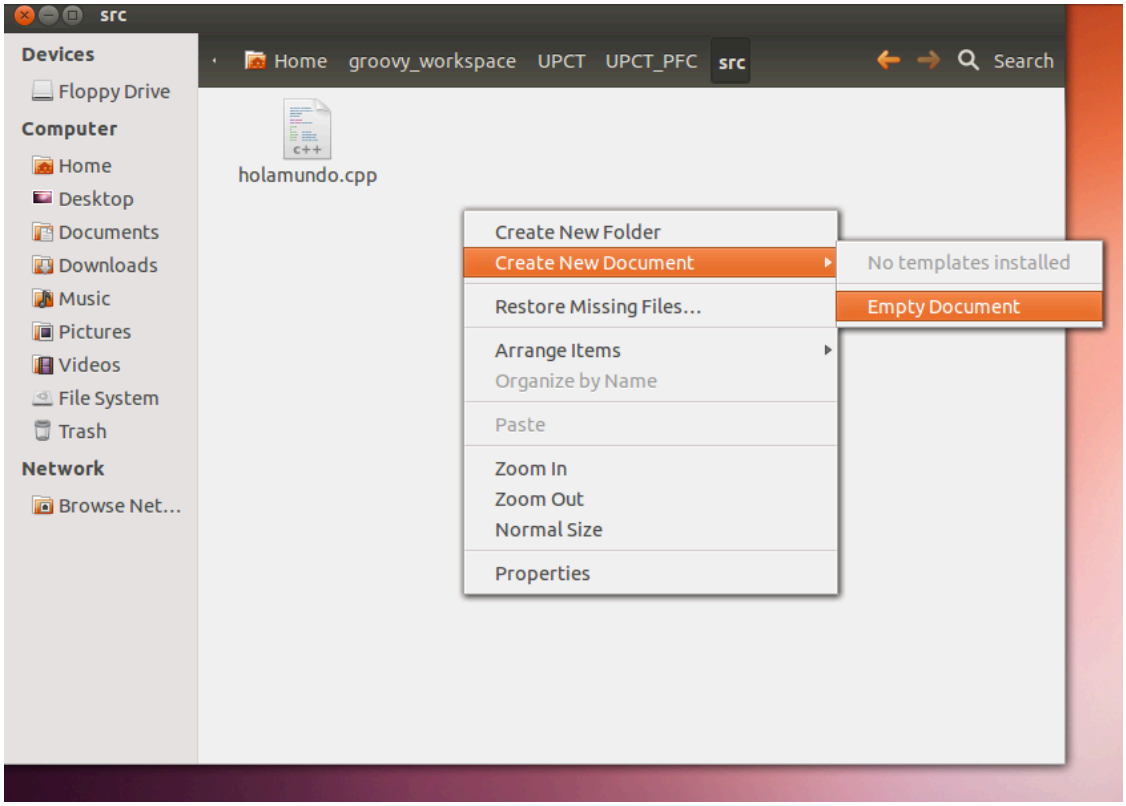


Imagen creando nuestro fichero .cpp

A continuación abrimos el fichero que acabamos de crear con un programa de edición de texto, o bien usando gedit tal y como lo hemos usado en el apartado anterior, y copiamos el siguiente código:

```
#include <ros/ros.h>

int main (int argc, char** argv)
{
  ros::init(argc, argv, "nodo_hola_mundo");//nombre del nodo
  ros::NodeHandle nh;
 //muestra por pantalla lo que hay dentro de las comillas
```

```
  puts("HOLA MUNDO");
  puts("-------------------------");
  puts("PFC Alvaro Garcia");

  ros::spin();
  return 0;

//espera en funcionamiento hasta que se lo indiquemos por teclado
}
```

Tras introducir el código, lo guardamos y lo cerramos, y ahora procedemos a editar el fichero CMakeList.txt y añadimos al final de todo el código:

rosbuild_add_executable(hola_mundo src/holamundo.cpp)

Esto hace que cuando generemos nuestro programa el sistema generará un ejecutable con el nombre hola_mundo, y cuyo código estará en la carpeta src/holamundo.cpp

Una vez realizado esto, debemos generar nuestro programa, para ello introducimos en nuestro terminal:

$ make

Lo que nos tras unos segundos nos devolverá una salida por pantalla parecida a esta

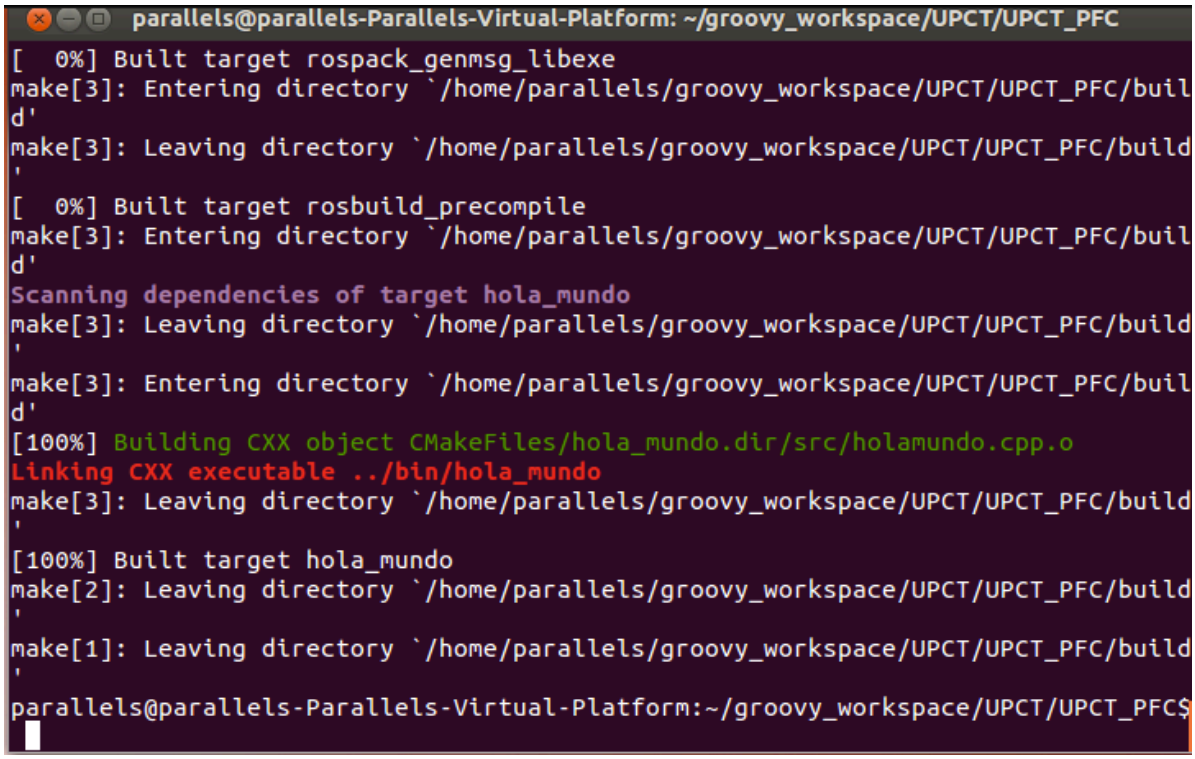


Imagen de ROS compilando nuestro programa

Y ya tenemos nuestro programa creado y listo para ser usado.

9.6 Arrancando nuestro programa

Una vez ya tenemos nuestro programa creado vamos a proceder a probarlo, para ello lo que hacemos es llamar a nuestro nodo tal y como hemos enseñado dentro del apartado correspondiente

Lo primero que hay que hacer es llamar a nuestro nodo master, es decir, abrimos un nuevo terminal y ejecutamos

$ roscore

Y ahora volvemos a nuestro terminal primero en introducimos la función

$ rosrun UPCT_PFC hola_mundo

Tras lo que veremos una salida por pantalla parecida a esta

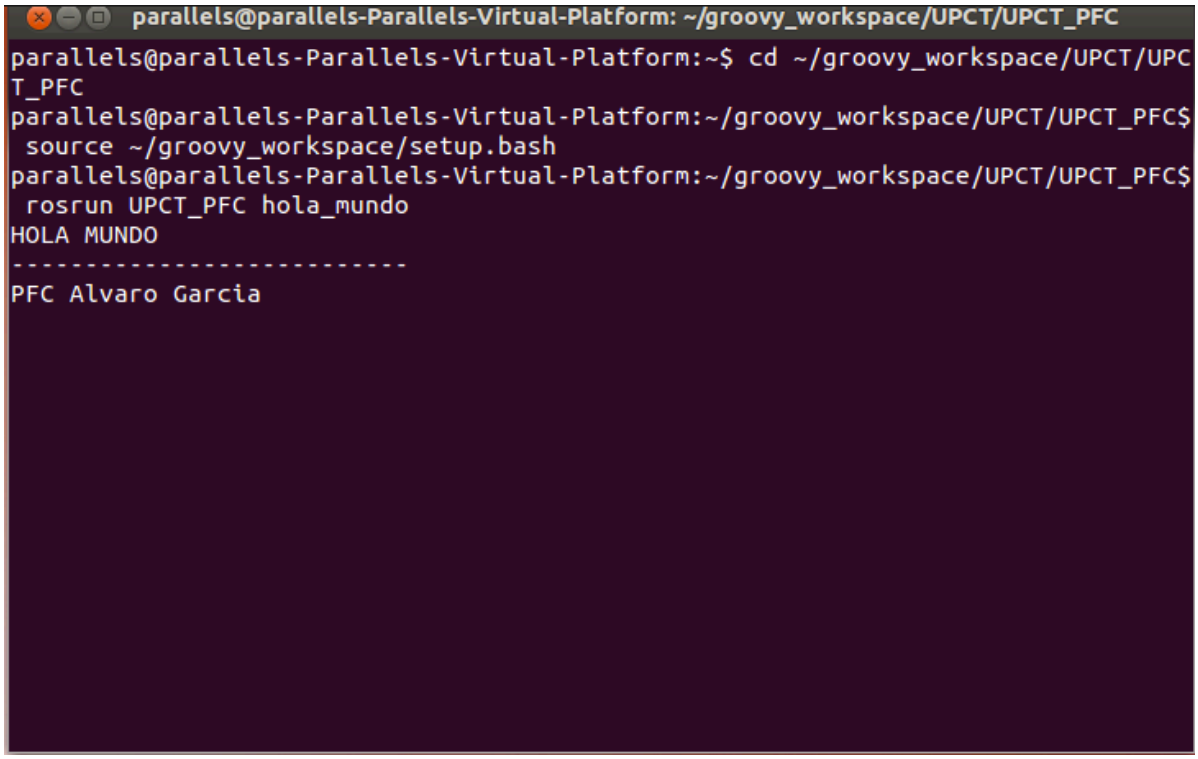


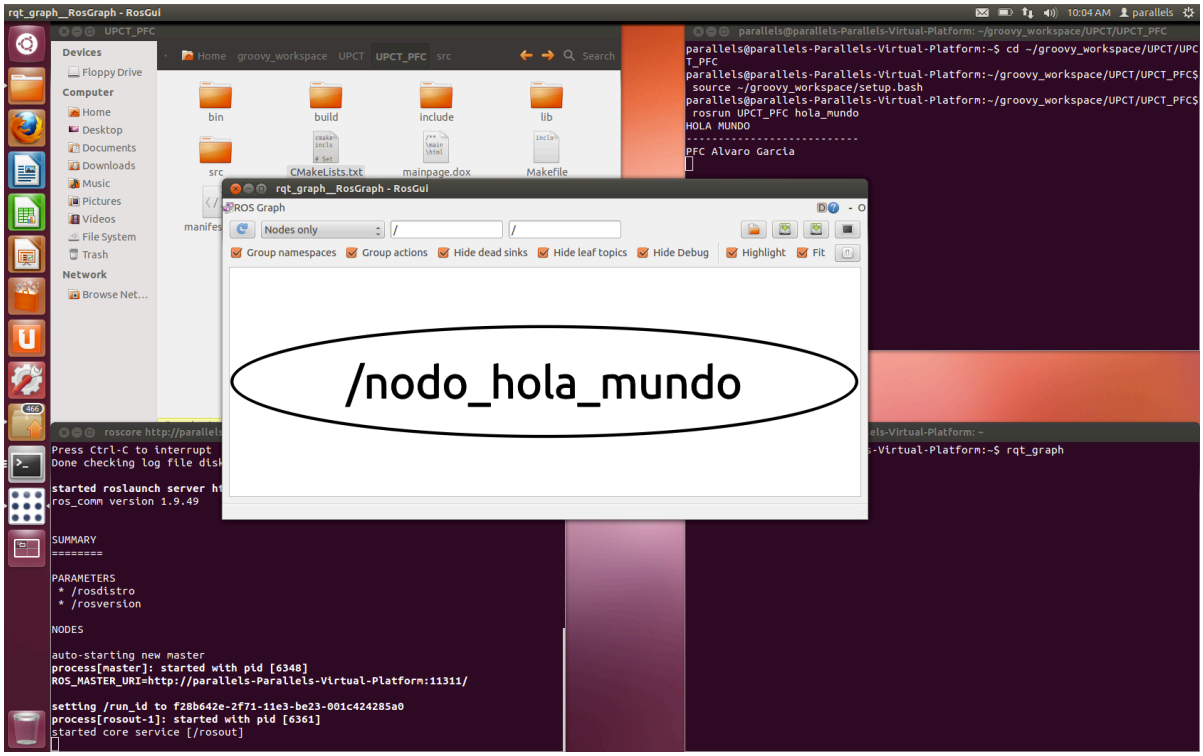
Imagen de nuestro programa funcionando.

En el caso que no se inicie el programa, comprueba que tienes en una pantalla de terminal diferente activado roscore, y que en la pantalla con la que estas trabajando has ejecutado con anterioridad la función

$ source ~/groovy_workspace/setup.bash

Ahora vamos a comprobar de forma gráfica que nuestro nodo está en funcionamiento, para ello abrimos una nueva pantalla de terminal y ejecutamos

$ rqt_graph



Representación gráfica de nuestro nodo.

10. Conclusión y proyectos futuros

Como hemos podido ver a lo largo de todo el documento, el sistema ROS es un sistema robótico muy completo que nos permite utilizar tanto el código antiguo de otros proyectos, como uno nuevo generado directamente para nuestro sistema, además de contar con soporte para una gran multitud de robots y sensores existentes en el mercado, y otra de las ventajas de las que se ha hablado es de una gran comunidad que se está dedicando a mejorar día a día el sistema, así como de ampliar las compatibilidades tanto con sistemas nuevos (como puede ser Raspberry Pi), como con los últimos sensores y actuadores disponibles en el mercado.

ROS es un sistema que en la actualidad cuenta con una gran cantidad de personas a nivel mundial que se dedican al mantenimiento y creación de nuevos repositorios, así como a la mejora del propio sistema, provocando una rápida adaptación a cualquier cambio.

Respecto a la compatibilidad, ROS es casi totalmente compatible con repositorios antiguos, lo que nos permite utilizar la programación que ya tenemos de otros sistemas y portarlos a este, permitiéndonos mover esta programación a casi cualquier máquina, y aquí es donde podemos encontrar un el primer pero, ya que aunque la comunidad está trabajando fuertemente que se pueda utilizar en cualquier ordenador, a día de hoy solo pueden asegurar el correcto funcionamiento de ROS en sistemas GNU (Linux) y MAC OS X, y se están encontrando con algunos fallos de programación en sistemas Windows y

Raspberry Pi, este último debido a que es un sistema totalmente nuevo y aún poco extendido.

Cabe destacar que un sistema que esta soportado por una gran comunidad tiene infinidad de cosas positivas, pero también tiene varios apartados negativos, el primero de ellos es que al existir gran cantidad de repositorios, no todos tienen por que funcionar totalmente, y la solución a los errores de estos dependen también de la comunidad, lo que puede hacer que si es un repositorio que no sea usado normalmente, el fallo no sea descubierto de inmediato, y la corrección tarde en llegar; y otro de los principales problemas que puede suceder, es que hoy en día cualquier fundación o sistema de este tipo necesita unos recursos bastante abundantes que son muy difíciles de cubrir con donaciones, y con pequeñas contribuciones por parte de empresas privadas, lo que puede llevar a que en épocas de crisis o con una competencia fuerte las empresas y socios prefieran invertir ese dinero en otros sistemas, pudiendo llegar a la descontinuación de ROS provocando la migración obligatoria a otros sistemas.

Como futuros proyectos propuestos se podría dividir en dos vertientes, una mas teórica, en la cual se pueden desarrollar robots en los entornos virtuales descritos en este documento, e incluso generar un Robot de los disponibles en los laboratorios, para comprobar que todo el programa funcionan de forma correcta,y una vertiente real, donde se vaya utilizando ROS para generar el código necesario para mover un robot real, al que se le pueden ir añadiendo mas funciones con el paso del tiempo. Por supuesto estas dos vertientes pueden ser complementaria, ya que lo ideal sería generar un robot en el sistema virtual, probar que todo el código es completamente funcional, y luego cargarlo en nuestro robot real.

11. Bibliografía

www.wikipedia.com

www.ros.org

wiki.ros.org

www.willowgarage.com

www.bipedolandia.es

http://www.generationrobots.com/en/content/55-ros-robot-operating-system

http://geeksroom.com/2013/08/ros-el-estandar-de-facto-de-la-industria-de-los-robots/78002/

http://readwrite.com/2013/05/09/how-an-open-source-operating-system-jumpstarted-robotics-research#awesm=~ogKpiH3BrKRskJ

http://robociencia.com/introduccion-a-ros/

https://moodle2012-13.ua.es/moodle/mod/wiki/view.php?id=23601

http://robociencia.com/ros-el-futuro-de-la-robotica-autonoma/

http://www.bipedolandia.es/t1654-explicacion-detallada-de-50-minutos-del-funcionamiento-de-kinect-y-ros-robot-operating-system

http://www.automatizar.org/2011/09/creando-aplicaciones-para-robots.html

http://www.instructables.com/id/Getting-Started-with-ROS-Robotic-Operating-Syste/

ROS By Example GROOVY - Volume 1 Autor: R. Patrick Goebel

http://es.wikipedia.org/wiki/Robótica