Guía de ROS

Gustavo Velasco-Hernández

3 de julio de 2014

Índice general

Capítulo 1

Introducción a ROS

ROS, (Robot Operating System) es una herramienta para la creación de aplicaciones en robótica cuyo objetivo es promover el reuso de código de diferentes autores y para diferentes aplicaciones. Qué es ROS exactamente, cuál es su finalidad, para qué sirve y quiénes lo usan actualmente son preguntas que se responderan en esta sección[1].

1.1. ¿Qué es ROS?

ROS es un "meta-sistema operativo" de código libre que provee diferentes servicios que se esperarían de un sistema operativo, como abstracción de hardware, control de dispositivos en bajo nivel, funciones que se usan comúnmente, comunicación de procesos mediante mensajes y administración por paquetes. Bajo la licencia BSD es de libre uso, modificación y comercialización.

También incluye herramientas y librerías que permiten obtener, construir, escribir y ejecutar programas entre varios computadores. ROS es similar en algunas cosas con otros frameworks usados en robótica, tales como Player, Orocos, Yarp, Orca, Microsoft Robotics Studio, entre otros. ROS no es un framework en tiempo real, aunque sí es posible integrarlo con código en tiempo real, por ejemplo el robot PR2 de Willow Garage usa un sistema que se encarga de transportar mensajes en tiempo real. Incluso se ha integrado con el toolkit de tiempo real de OROCOS.

1.2. Objetivos de ROS

El objetivo de ROS no es ser un framework con las mejores características, en vez de esto, el objetivo principal de ROS es permitir el reuso de código en la investigación y el desarrollo de la robótica. ROS es un framework distribuido

[1]Este capítulo está basado en la documentación de ROS en http://wiki.ros.org/ROS/introduction

de procesos (llamados nodos) que permiten ser diseñados y acoplados de una manera sencilla en tiempo de ejecución. Estos nodos son organizados en paquetes, con lo que pueden ser fácilmente compartidos y distribuidos. ROS también tiene un sistema de repositorios que permiten la colaboración y distribución del código. Con este diseño, desde el sistema de archivos hasta la comunidad, se independiza el desarrollo y la implementación. Otros sub-objetivos en pro de cumplir con el principal ideal son:

- Ser un sistema ligero y sencillo, integrable con otros frameworks
- Desarrollo de librerías agnósticas
- Independiente del lenguaje (Python, C++, Lisp, Java, Lua)
- Fácil de probar
- Escalable

1.3. ¿Qué contiene ROS?

Entre el conjunto de librerías y herramientas que contiene ROS, cabe destacar su integración con software de procesamiento de imágenes como OpenCV[2] y para el procesamiento de nubes de puntos como PCL[3]. ROS también se integra con plataformas de simulación 2D como Stage[4] y simulación 3D como Gazebo[5].

Incluido en ROS se encuentra RVIZ que es una herramienta de visualización 3D que permite integrar información de sensores, modelos de robots y otros datos en una sola vista y también permite interactuar con esta visualización.

También ROS incluye rqt, un framework para desarrollo de interfaces gráficas basadas en QT. Estas interfaces son desarrolladas en forma de plugins, permitiendo que se puedan ejecutar individualmente u organizadas en una misma ventana. Estos plugin se pueden usar para la visualización de variables, gráficos, monitoreo de mensajes, nodos, entre otras cosas.

Otras utilidades que posee ROS son las herramientas de consola, que son un conjunto de comandos que sirven, entre otras cosas, para la navegación entre archivos y paquetes de ROS, ejecución de aplicaciones, creación de paquetes, etc.

1.4. ¿Por qué usar ROS?

El desarrollo en robótica se sustenta en otros subcampos como navegación, percepción, planeación, razonamiento, entre otros. Debido a la amplitud de estos

[2]Open Source Computer Vision (http://opencv.org/)
[3]Point Cloud Library (http://pointclouds.org/)
[4]Stage Robot Simulator (http://rtv.github.io/Stage/)
[5]Gazebo Simulator (http://gazebosim.org/)

temas, la experiencia necesaria para programar un robot va más allá de la capacidad de una sola persona. Por esto se hace necesario simplificar la integración de diferente software de diferentes orígenes.

Además la robótica es un problema de integración de sistemas complejos y requiere un conjunto de herramientas para manejar esta complejidad. También es necesario que todos sus componentes puedan comunicarse entre sí de una manera eficiente ya que estos operan en el mundo real. ROS ofrece enfoques para cada uno de estos desafíos.

A Continuación se presentan algunas ventajas y desventajas del uso de ROS[6]

Ventajas

- Gran cantidad de material desarrollado para robótica.
- Hay una gran comunidad activa con interés en ROS.
- Además de robótica, hay otros campos incluidos como ingeniería de software, colaboración en el desarrollo, modelos de código abierto, etc.
- Accesible para todos, no solo expertos en ciencias de la computación.
- Sirve de contexto para muchos sub-problemas de la robótica, así se puede observar como todo funciona en conjunto
- Es una buena herramienta para problemas de M.Sc y Ph.D y es usado activamente en investigación
- Amplio soporte de hardware (y creciendo).
- Amplio soporte para simulación.
- El reuso de código hace que lo que se demoraba hace unos años 3 meses, hoy toma un día.

Desventajas

- La curva de aprendizaje, especialmente para personas no relacionadas con las ciencias de la computación. Es un framework muy grande y puede ser aterrador. También puede ser difícil encontrar las cosas que se necesitan.
- Linux, línea de comandos, multiterminal, multiproceso, distribuido, asíncrono. Puede ser más aterrador.
- Gran cantidad de material desarrollado para robótica. Cuidado con los instructores perezosos, esto puede ser una desventaja.
- Poco material dirigido al entorno universitario (Por ahora).
- Falta de una comunidad cohesiva de educadores de ROS (Por ahora).

1.5. Ejemplos del uso de ROS

Debido a que ROS fue concebido como una herramienta para realizar aplicaciones en el campo de la robótica, su uso se ha extendido en todas las áreas y escenarios concernientes a esta. Se usa tanto en manipuladores como en plataformas móviles. En universidades y entornos académicos, ya sea para enseñanza o investigación y también en la industria. Incluso se creó el proyecto ROS Industrial[6] como una extensión de ROS para aplicaciones industriales.

ROS también es un sistema multiplataforma; se ha usado con robots humanoides como NAO, plataformas para investigación como el PR2, robots industriales como Baxter, vehículos como los robots Pioneer, Kuka o Turtle Bot; e incluso plataformas educativas como el Lego Mindstorm NXT y la computadora Raspberry Pi. Algunas de estas plataformas se muestran en la figura 1.1.

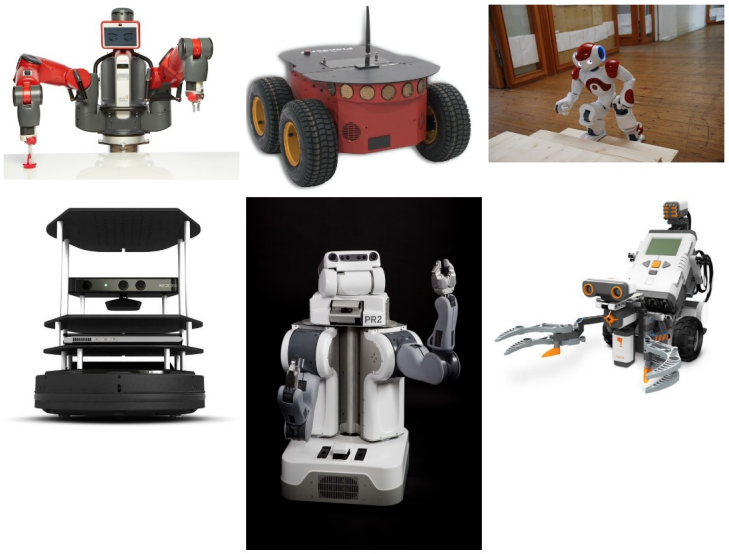

Figura 1.1: Algunas plataformas en las que se usa ROS. Desde arriba a la izquierda: Baxter, Pioneer AT, NAO, TurtleBot 2, PR2, Lego Mindstorm NXT

[6]ROS Industrial (http://rosindustrial.org/)

Capítulo 2

Conceptos básicos en ROS

Como ROS es un sistema tan grande y complejo, se puede dividir en tres niveles para su mejor comprensión y apropiación. El primero es el nivel del sistema de archivos el cual describe la forma en cómo ROS trabaja con los paquetes, su definición y los archivos y carpetas que estos contienen. El segundo nivel es el nivel funcional, llamado grafo de computación, en el que se especifican los elementos que hacen parte de la ejecución, su relación y su arquitectura de comunicaciones. El tercer y último nivel es el nivel de la comunidad, lo que incluye el manejo del software en cuanto a la contribución de paquetes, las distribuciones, los repositorios, las listas de correo, entre otros.

2.1. Sistema de archivos

El sistema de archivos consiste en todos los recursos que se encuentran en disco después de instalarse ROS. Esto comprende los archivos y la estructura de directorios.

2.1.1. Conceptos Básicos

Los elementos que componen este sistema son los paquetes (packages), los meta-paquetes (Meta-packages), el manifiesto de paquete (Package Manifest), los tipos de mensaje (Message type) y los tipos de servicio (Service type).

Paquetes Los paquetes son la principal estructura organizacional de ROS. Puede contener procesos (nodos), librerías, archivos de configuración, datasets y demás elementos que al organizarse juntos sean de utilidad.

Meta-Paquetes Los Meta-paquetes son paquetes creados especialmente para agrupar paquetes que comparten cierta similitud. Comunmente son utilizados como una herramienta de compatibilidad para un elemento de versiones anteriores, los Stacks.

Manifiesto de Paquete Los manifiestos son archivos .xml que proveen información de los paquetes, por ejemplo datos como nombre, versión, autores, licencias, dependencias, entre otros.

Tipos de Mensajes Son archivos .msg que contienen la descripción de la estructura de datos para el envío de cierto tipo de mensaje.

Tipos de Servicios Son archivos .srv que contienen la descripción de la estructura de datos para las peticiones y respuestas de determinado tipo de servicio.

2.1.2. Estructura de Directorios

En disco, cada paquete corresponde a un directorio con el mismo nombre. Comunmente este directorio contiene dos archivos: package.xml y CMakeLists.txt. Además, contiene directorios como include, src, msg, srv y scripts.

El archivo package.xml es el manifiesto del paquete que, como se mencionó anteriormente, contiene información sobre el paquete. El archivo CMakeLists.txt es usado para la compilación del código fuente y por lo general es autogenerado. Los directorios include y src son los que contienen el código de los procesos o nodos implementados, usualmente en C++. Por otra parte, el directorio scripts contiene archivos ejecutables, ya sean nodos o archivos con otra utilidad, en lenguaje interpretado como Python o bash. Finalmente los directorios msg y srv contienen los archivos que describen los tipos de mensajes y tipos de servicios, respectivamente.

2.2. Grafo de computación

El funcionamiento de ROS se basa en una arquitectura de comunicación peer-to-peer, entre unidades llamadas nodos, los cuales se comunican entre sí intercambiando información. Para entender un poco más y profundizar en el funcionamiento de ROS es necesario describir algunos conceptos de este sistema, que son nodos (nodes), maestro (master), servidor de parámetros (parameter server), mensajes (messages), temas (topics) y servicios (services). A este conjunto de elementos junto a la arquitectura de comunicación se les llama grafo de computación (Computation graph).

Nodos Los nodos son los procesos que se ejecutan, estos se comunican entre sí usando los temas para la transmisión de los mensajes, usando invocación de servicios y usando el servidor de parámetros. Los nodos están pensados para ser granulares, es decir, un sistema para controlar un robot suele posee muchos nodos; Un nodo puede controlar el sensor láser, otro nodo controla los motores, otro nodo puede realizar la localización, otro nodo realiza la planeación de rutas y otro nodo puede controlar la representación gráfica del sistema.

Maestro El maestro ROS (ROS master) es el que provee al sistema de servicios de nombramiento y registro a los nodos, además permite la búsqueda de los elementos en el grafo de computación. El maestro también realiza el seguimiento de los publicadores y suscriptores a los temas, así como de los servicios. El papel del maestro es permitir a los nodos encontrarse el uno al otro y una vez localizados, estos se pueden comunicar entre sí.

Servidor de Parámetros El servidor de parámetros hace parte del maestro y se usa para almacenar datos en una ubicación central, accesible por los nodos en tiempo de ejecución. Usualmente es utilizado para almacenar valores estáticos y parámetros de configuración.

Mensajes Los mensajes son datos que con los que se comunican los nodos entre sí. Estos mensajes son como estructuras similares a las usadas en lenguaje C, que contienen campos tipados. Tambien los campos pueden ser arreglos de estos tipos y otros mensajes.

Temas Los temas son los encargados de enrutar los mensajes. Pueden ser vistos como el canal por el cual viajan los mensajes entre los nodos. Estos pueden suscribirse a los temas y también publicar en ellos; además un nodo puede publicar y suscribirse a varios temas y al mismo tiempo, varios nodos pueden publicar y suscribirse a un tema.

Servicios Aunque los temas ofrecen flexibilidad en la comunicación, no es apropiado para interacción de tipo petición-respuesta, algo que puede ser requerido en un sistema distribuido. Este tipo de interacción puede implementarse usando los servicios. Estos se definen como una estructura con un par de mensajes, uno para la petición y otro para la respuesta. Cada servicio se registra con un nombre en un nodo para hacerlo disponible.

2.3. Comunidad

El nivel de la comunidad incluye los elementos que tienen que ver con el intercambio de software y conocimiento sobre ROS. Estos elementos comprenden las distribuciones, los repositorios, la wiki, la lista de correo y el sitio de preguntas.

Las distribuciones de ROS, son un conjunto de paquetes versionados que pueden ser instalados. Similar a las distribuciones de los sistemas Linux, hacen posible installar un conjunto de software y hacerlo sostenible. Los repositorios son los lugares donde se almacena el código fuente, donde diferentes instituciones o personas comparten el código desarrollado y es liberado.

Por otra parte, la wiki es la principal fuente de información sobre ROS y sus componentes. Contiene la documentación y cualquiera puede publicar en ella, hacer correcciones o actualizaciones y escribir tutoriales. La lista de correo es el medio más rápido para enterarse de las noticias de ROS; nuevo software, actualizaciones, eventos, Bugs, etc. Finalmente, el sitio de preguntas, answers.ros.org es el lugar principal para la solución de dudas.

Capítulo 3

Instalando ROS

Las instrucciones que se presentan a continuación, son para instalar ROS Hydro Medusa en Ubuntu 12.04 (Precise Pangolin). Esta distribución de ROS solo es compatible con las distribuciones de Ubuntu 12.04 (Precise), 12.10 (Quantal) y 13.04 (Raring).

3.1. Configuración inicial de los repositorios

Para empezar debemos configurar Ubuntu, específicamente la utilidad apt, para que reconozca los repositorios de ROS y nos permita instalarlo y actualizarlo. Por defecto en Ubuntu 12.04 (precise) ya están habilitados los repositorios universe, multiverse y restricted, lo cual era un paso adicional cuando se instalaba ROS en versiones anteriores de Ubuntu. Para verificarlo podemos abrir la interfaz gráfica de orígenes de software con el siguiente comando:

```
$ python /usr/bin/software-properties-gtk
```

Aquí podemos verificar que están seleccionados los cuatro tipos de repositorios (main, universe, restricted y multiverse). Ahora, lo que debemos hacer es añadir los repositorios de ROS a la lista de fuentes de software y configurar el acceso con los siguientes comandos.

```
$ sudo sh -c 'echo "deb␣http://packages.ros.org/ros/ubuntu␣precise␣main
    " > /etc/apt/sources.list.d/ros-latest.list'
$ wget http://packages.ros.org/ros.key -O - | sudo apt-key add -
```

Ahora ya tenemos los repositorios de ROS configurados para su administración desde el comando apt.

3.2. Instalación y configuración inicial

Aunque ya configuramos los repositorios de ROS para su funcionamiento con apt, debemos corroborar que el listado de paquetes esté actualizado:

```
$ sudo apt-get update
```

Debido a que ROS contiene gran cantidad de herramientas y utilidades, hay 4 opciones de instalación. La primera, la recomendada y la que vamos a usar, es la instalación completa (Desktop-full), la cual incluye todo el core de ROS, librerías genéricas de varios robots, simuladores 2D y 3D, herramientas de visualización y librerías de navegación y percepción.

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

La segunda opción (Desktop) incluye el core, las librerías de robots y las herramientas de visualización.

```
$ sudo apt-get install ros-hydro-desktop
```

La tercera opción (ROS-base) solamente tiene el core de ROS y librerías de comunicación, nada de interfaces gráficas (GUI).

```
$ sudo apt-get install ros-hydro-ros-base
```

La última opción es instalar paquetes individuales especificando el nombre en el comando:

```
$ sudo apt-get install ros-hydro-PACKAGE
```

por ejemplo:

```
$ sudo apt-get install ros-hydro-slam-gmapping
```

Para la instalación de la versión completa se descargan aproximadamente 900 MB, que descargando a 1M/s se demoraría unos 15 minutos, y tras esto se configurarán los paquetes descargados, así que mientras se descarga, podemos ir por un café y leer un poco sobre la estructura y el funcionamiento de ROS si no lo hemos hecho :D

...

Después de haber finalizado la instalación completa, debemos configurar algunas herramientas adicionales y configurar nuestro entorno de trabajo.

Empezaremos inicializando rosdep, una herramienta que nos facilita la instalación de algunas dependencias de paquetes y que es necesaria para ejecutar algunos componentes de ROS

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

Es conveniente que cada vez que iniciemos sesión las variables de entorno de ROS sean cargadas automáticamente:

```
$ echo "source␣/opt/ros/hydro/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
```

Verificamos que ya están declaradas las variables de entorno de ROS:

```
$ export | grep ROS
declare -x ROS_DISTRO="hydro"
declare -x ROS_ETC_DIR="/opt/ros/hydro/etc/ros"
declare -x ROS_MASTER_URI="http://localhost:11311"
declare -x ROS_PACKAGE_PATH="/opt/ros/hydro/share:/opt/ros/hydro/stacks
    "
declare -x ROS_ROOT="/opt/ros/hydro/share/ros"
```

Ahora instalaremos rosinstall, una herramienta que nos ayudará a descargar las fuentes de los paquetes y a instalarlos. Se descargarán aproximadamente 10 MB.

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

Finalmente, tenemos ROS Hydro Medusa instalado y configurado en nuestro sistema Ubuntu Precise Pangolin.