TALLER

Robot

Operating

System

https://www.ros.org/

INTRODUCCIÓN

ROS es una plataforma software para robótica de código abierto, que se emplea tanto para robots comerciales como para investigación. Proporcionando ciertas características para evitar que cualquiera que empiece a desarrollar el software de un robot, tenga que empezar de cero. Como veremos, en si, ROS pese a su nombre, no es un sistema operativo, si no un meta sistema operativo que provee ciertas funcionalidades. Por este motivo ha de instalarse sobre otro sistema operativo y así ROS extiende su funcionalidad adaptándolo a aplicaciones robóticas. Para esto dispone de herramientas de desarrollo, visualización, depuración, etc.
Es por eso por lo que ROS y Linux van de la mano, cada versión de ROS está asociada a una distribución de Linux, más concretamente, a una versión de Ubuntu que es la distribución que soporta completamente ROS. Por lo que debemos tener unas habilidades mínimas en GNU/Linux para poder trabajar con ROS. A nivel de programación, en ROS, principalmente se emplean dos lenguajes C++ y Python.

Se facilita una máquina virtual con Linux+ROS, más concretamente Ubuntu 20.04 LTS + ROS Noetic
Password: huro

https://unialicante-my.sharepoint.com/:u:/g/personal/jl_ramon_mscloud_ua_es/EaLXmGzIZ6xHifj6EtbU4XgB0seC68zsrrA5oIZ9t8HIew?e=sIS01d

Otras maquinas virtuales de ROS Industrial
Password: rosindustrial
https://rosi-images.datasys.swri.edu/

Además, en este enlace puedes encontrar una guía de cómo realizar la instalación http://wiki.ros.org/noetic/Installation/Ubuntu.
El objetivo es que el alumno entienda los conceptos básicos de la plataforma para robótica ROS.

ROS

ROS Arquitectura y conceptos.

Para entender mejor cual es la arquitectura de ROS, vamos a ejecutar un pequeño ejemplo que nos ayudará a la hora de explicar ciertos conceptos.
Vamos a abrir una terminal e introduciremos el siguiente comando:

```
$roscore
```

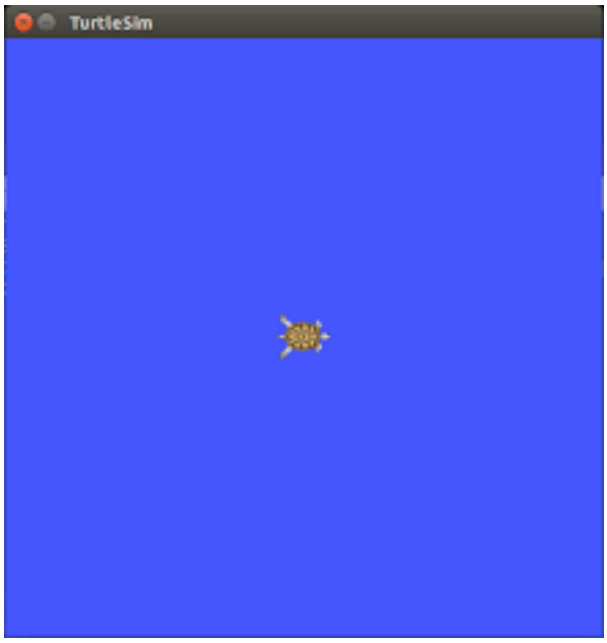
Si todo es correcto acto seguido tiene que arrancar en nodo principal de ROS mostrando …

```
... logging to /home/huro/.ros/log/49d6af12-f1be-11e9-8925-080027131460/roslaunch-mayr-
6214.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://mayr:40629/
ros_comm version 1.12.14


SUMMARY
========

PARAMETERS
 * /rosdistro: kinetic
 * /rosversion: 1.12.14

NODES

auto-starting new master
process[master]: started with pid [6225]
ROS_MASTER_URI=http://mayr:11311/

setting /run_id to 49d6af12-f1be-11e9-8925-080027131460
process[rosout-1]: started with pid [6238]
started core service [/rosout]
```

En estos momentos está corriendo el nodo maestro, sin cerrar esta terminal abriremos otra nueva e introduciremos:

```
$ rosrun turtlesim turtlesim_node
```

Una vez más si todo es correcto aparecerá una ventana con una tortuga en el centro.



Y este será el “Hola Mundo” de ROS, pero vamos a aprovechar para explicar la arquitectura de ROS en base a este ejemplo. ¿Qué es lo que ha sucedido? Primero hemos ejecutado una instrucción que ha iniciado el nodo maestro. Éste tiene que estar siempre ejecutándose mientras queramos que funcione nuestro robot. El maestro queda a la escucha hasta que otro nodo se inicia, que es lo que hemos hecho en la segunda terminal. Cuando se inicia un nodo cualquiera, se registra en el nodo maestro indicando toda la información que puede resultar útil para el resto de los nodos. Como hemos comentado antes, la piedra angular de ROS es facilitar la comunicación entre nodos y esto es así porque dentro del software de control de un robot recibiremos información de múltiples sensores, aplicaremos un control y enviaremos información a los actuadores. ¿Quiere decir esto que todas las comunicaciones pasan por el nodo maestro? No, el nodo maestro no está en medio de todas las comunicaciones pudiendo convertirse en un cuello de botella, si no que hace la función de una telefonista. Inicialmente contactas con el nodo maestro para conocer dónde se encuentra el nodo con el que vamos a comunicar y una vez establecida la comunicación está se realiza entre nodos. Otra gran ventaja que radica en esta arquitectura es que la aplicación puede estar distribuida en distintas máquinas siempre que no exista más de una en la que se ejecuta el nodo maestro.
Resumiendo, ROS estará compuesto por los siguientes elementos:

Master

El maestro será donde se registran los nodos y donde acuden a buscar a otros nodos.

```
$ roscore
```

Nodes

Son los procesos que realizan las distintas tareas necesarias para nuestro robot, como por ejemplo leer información de un sensor de distancias laser o enviar una referencia de velocidad a una rueda de nuestro robot.

```
$ rosrun package_name node_name
```

Ejecuta node_name del paquete package_name.

```
$ rosnode list
```

Muestra un listado de los nodos activos.

```
$ rosnode info node_name
```

Muestra información del nodo node_name.

Parameter Server

El servidor de parámetros permite el almacenamiento de valores en base a una clave de forma centralizada y común a todos los nodos. El “Parameter Server” utiliza YAML como leguaje de marcado.

```
$ rosparam list
```

Muestra un listado de los parámetros.

```
$ rosparam set param_name value
```

Guardar un valor en los parámetros

```
$ rosparam get param_name
```

Nos muestra en valor de un parámetro.

Messages

Los nodos se comunican entre ellos pasándose mensajes. Un mensaje es simplemente la definición de una estructura de datos para modelar la información que compartirá. La descripción se guarda en un archivo .msg.

Topics

Una de las formas de comunicarse entre nodos es mediante “Topics”. Un Topic es el nombre que recibe el canal de comunicación. Los nodos pueden publicar o subscribirse a un Topic, normalmente suele ser una arquitectura 1 publica N suscriben.

```
$ rostopic list
```

Muestra una lista de los Topics activos.

```
$ rostopic echo /Topic
```

Se suscribe y muestra el contenido de Topic.

```
$ rostopic info /Topic
```

Muestra información del Topic.

Service

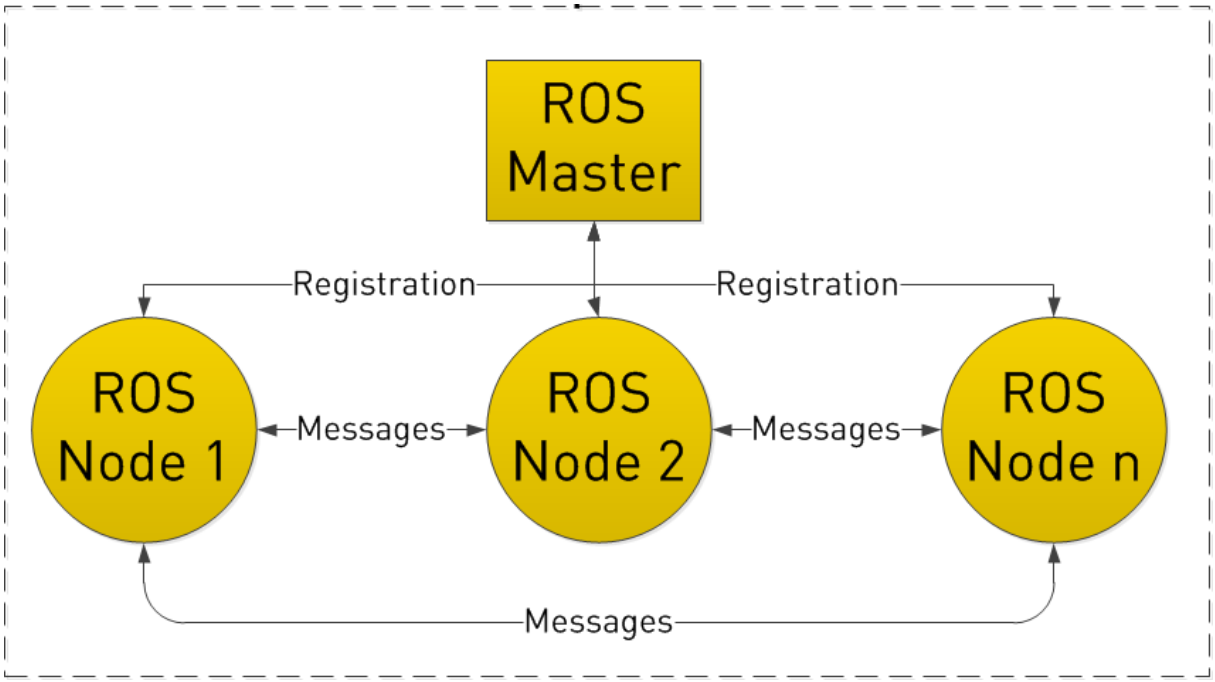
Los servicios son la otra forma que tiene ROS de comunicar entre nodos. Pero en los servicios el nodo puede enviar una petición y recibir una respuesta.

```
$ rosservice list
```

Lista los servicios activos.

```
$ rosservice type service_name
```

Muestra de qué tipo es el servicio.



Hemos visto cuales son los principales aspectos de la arquitectura de ROS en su funcionamiento. A nivel de desarrollo en ROS el código se organiza en paquetes y metapaquetes. Un paquete puede contener uno o varios nodos, más el software necesario que facilite el uso de estos módulos por terceros. Además, debería cumplir la norma de implementar la suficiente funcionalidad como para ser útil, pero evitar ser demasiado pesado. La creación y compilación de estos paquetes se gestiona con la herramienta Catkin. Catkin son una serie de macros de compilación a bajo nivel para ROS. Para poder trabajar en ROS con Catkin tenemos que crear un espacio de trabajo, que será el que albergue los paquetes que estamos desarrollando. Para crear un espacio de trabajo ejecutaremos estos comandos;

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin_make
$ source devel/setup.bash
```

Esto tiene que haber creado la siguiente estructura de directorios:

Catkin_ws/
- build/
- devel/
- src/
- install

De todas estás carpetas ahora mismo nos vamos a centrar en “src” que es donde crearemos todos nuestros paquetes. Así, tras asegurarnos que estamos dentro de “catkin_ws/src/” cuando queramos crear un nuevo paquete ejecutaremos:

```
$ catkin_create_pkg mi_primer_package std_msgs rospy roscpp
```

Ahora tenemos que crear el launch_file. Un launch_file es una herramienta que empleamos para cuando tenemos que iniciar nodos con parámetros o de forma coordinada. Por lo que creamos una carpeta llamada launch dentro de la carpeta de nuestro paquete al mismo nivel que el src donde hemos puesto nuestro código fuente.

```
$ roscd mi_primer_package
$ mkdir launch
```

Dentro crearemos un archivo con el nombre mi_primer_package_launch_file.launch con el siguiente contenido:

```
<launch>
  <node name="talker" pkg="rospy_tutorials" type="talker" />
</launch>
```

Para ejecutar nuestro nodo utilizaremos la instrucción

```
$ roslaunch mi_primer_package mi_primer_package_launch_file.launch
```

Si todo es correcto en nuestra terminal debe aparecer algo parecido a lo siguiente:

```
catkin_ws$ roslaunch mi_primer_package mi_primer_package_launch_file.launch
... logging to /home/user/.ros/log/1f10268c-f36c-11e9-957b-0a26c7f36644/roslaunch-
rosdscomputer-25371.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://rosdscomputer:34223/

SUMMARY
========

PARAMETERS
 * /rosdistro: kinetic
 * /rosversion: 1.12.14

NODES
  /
    SoyNodo (mi_primer_package/simple.py)

ROS_MASTER_URI=http://master:11311

(En esta parte aparece la información correspondiente a los nodos lanzados)
```

EJERCICIO 1

Para verificar que el alumno ha entendido todos los pasos para crear paquetes, nodos y ejecutarlos, accede a la siguiente dirección y realiza el tutorial de “Simple Publisher and Subscriber”

http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29

Para que sea una versión mejorada, el alumno ha de personalizar los mensajes añadiéndole un toque de originalidad al ejemplo, modificando los mensajes. Ha de entregarse la carpeta del paquete completa con todo lo necesario para poder ejecutarse.

EJERCICIO 2

El segundo ejercicio que vamos a realizar nos servirá para afianzar los conocimientos adquiridos. En este caso el alumno tiene que desarrollar los nodos que implementan la funcionalidad de un pequeño robot móvil que se encuentra en un punto de una habitación de paredes paralelas. Para la implementación han de crearse dos nodos. El primero de ellos da una medida de distancia del sensor a la pared o (-1) si el robot se encuentra girando. El segundo ha de suscribirse al nodo que realiza las medidas y cada cuatro medidas de distancia (distancia + giro x 4) mostrará como resultado cuál es el largo y ancho de la habitación, la superficie total y la posición del robot en coordenadas X, Y. Ha de entregarse la carpeta del paquete completa con todo lo necesario para poder ejecutarse.