

# Introducción a ROS/Stage

Martin Adams, Felipe I. Inostroza F., Daniel V. Lühr S.

<2014-03-20 jue>

## 1. Introducción a ROS

ROS<sup>1</sup> (Robot Operating System) es una infraestructura de software libre que facilita el desarrollo de software para robots. Provee servicios estándar de un sistema operativo, tales como abstracción del hardware, control de dispositivos, mensajería entre procesos, entre otros. Uno de sus componentes incluye el simulador robótico Stage<sup>2</sup> (originalmente parte del proyecto Player) que permite la simulación de múltiples robots móviles con sensores y objetos variados en un mundo virtual (de manera análoga al simulador KiKS utilizado en realizaciones anteriores de este curso).

### 1.1. Instalación

En esta actividad se utilizará ROS en su versión Groovy. La única plataforma para la cual existe una versión estable para ROS es Ubuntu, por lo que se recomienda altamente instalar alguna versión de dicho sistema operativo (u otra distribución de Linux) en el computador en que se va trabajar.

Alternativamente, es posible utilizar la máquina virtual<sup>3</sup>.

#### 1.1.1. Pasos de instalación

- Para instalar ROS en Ubuntu, abrir un terminal e introducir la siguiente instrucción:

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

- Una vez instalado, ejecutar:

```
$ roscd stage  
$ rosmake stage
```

- Para verificar la correcta instalación del programa, abrir un *core* de ROS mediante la instrucción:

```
$ roscore
```

- La ejecución de cualquier aplicación de ROS requiere que **roscore** esté inicializado. Cuando se cumpla esta condición, abrir otro terminal e introducir:

```
$ rosrun stage stageros /opt/ros/groovy/stacks/stage/world/willow-erratic.world
```

- Debería desplegarse una ventana con Stage mostrando el *mundo* introducido como argumento. Los archivos *.world* son la serie de parámetros de configuración de la simulación, entre ellos el nombre del escenario de simulación (generalmente una imagen png), las posiciones de los robots y la resolución espacial.

---

**Observación** la instrucción para ejecutar cualquier aplicación de ROS sigue la estructura:

```
$ rosrun [nombre_paquete] [nombre_del_ejecutable] [argumentos]
```

En el caso anterior el argumento fue el archivo *.world* que viene como ejemplo en el paquete stage.

---

<sup>1</sup><http://www.ros.org/>

<sup>2</sup><http://www.ros.org/wiki/stage>

<sup>3</sup>Ésta puede encontrarse en <http://wiki.ros.org/groovy/Installation>

## 1.2. Ejecución

Resulta curioso observar que en la simulación de ejemplo no se aprecia movimiento del robot, ya que no está recibiendo instrucción alguna. Sin embargo, en cualquier simulación robótica es necesario comandar los robots y realizar lecturas de sensor. En Stage esto puede llevarse a cabo de dos formas:

1. Con controladores de Stage: programas escritos en C++ que utilizan las funciones propias del simulador e independientes de ROS (Stage se incluyó como paquete de ROS en versiones recientes, pero existen versiones standalone con funciones propias).
2. Comunicar ROS con Stage mediante programas escritos en Python o C++ (que son los lenguajes para los que ROS provee una API). La ventaja de esto es que permite la comunicación con Stage mediante funciones genéricas de ROS, es decir, las mismas que ROS utilizaría para la comunicarse con un robot real o con otro simulador (como Gazebo). Esto permite que los programas escritos para Stage sean adaptables a otras plataformas.

En esta experiencia se utilizarán *scripts* en Python para comandar los robots, los cuales serán llamados controladores (no confundir con los controladores de Stage mencionados en el punto 1. A continuación se detallarán los pasos para la ejecución de un controlador.

### 1. Creación de un paquete ROS

Esta etapa sólo se realiza una vez. Cuando se creen nuevos controladores basta guardarlos en el paquete creado y ejecutarlos con las instrucciones de la Etapa 2 Ejecución. A continuación, los pasos a seguir:

- a) Crear un directorio para almacenar paquetes ROS propios. Tanto el directorio padre como el nombre del directorio creado pueden ser escogidos a gusto del alumno.

```
$ cd ~/
$ mkdir my_packages
```

- b) Ingresar al directorio recién creado y añadirlo a la variable de entorno ROS\_PACKAGE\_PATH (ruta donde ROS busca paquetes).

```
$ cd ./my_packages
$ export ROS_PACKAGE_PATH=~/.my_packages:$ROS_PACKAGE_PATH
```

- c) Crear paquete (de nombre arbitrario) mediante la instrucción:

```
$ roscreate-pkg EL5206 std_msgs rospy nav_msgs sensor_msgs
```

---

**Observación** la instrucción para crear una paquete de ROS sigue la estructura:

```
$ roscreate-pkg [nombre_paquete] [dependencias]
```

---

- d) Copiar las carpetas `scripts` y `worlds` presentes en `Tutorial.zip` en el directorio del paquete creado `my_packages/EL5206`

### 2. Ejecución

Para la ejecución de un controlador será necesario abrir tres terminales:

**Terminal 1** ejecución de roscore.

```
$ roscore
```

**Terminal 2** ejecución de Stage.

```
$ rosrunc stage stageros ~/my_packages/EL5206/worlds/world1.world
```

Donde `world1.world` puede ser cambiado por otro mundo.

**Terminal 3** ejecución de controlador.

Primero, es necesario asegurarse que la variable de entorno de ROS incluye la ruta al directorio del paquete creado, si no es el caso se debe incluir con:

```
$ export ROS_PACKAGE_PATH=~/.my_packages:$ROS_PACKAGE_PATH
```

Ahora, es posible ejecutar el controlador, en este caso el *script* `stage_controller.py`.

---

**Observación** los nuevos controladores deben ser guardados en el directorio `/scripts` y deben ser marcados como ejecutables.

Además, es necesario asegurarse que en el *script*, la función `roslib.load_manifest` recibe como argumento el nombre del paquete ROS.

---

Luego, introducir:

```
$ rosrun EL5206 stage_controller.py
```

---

**Observación** `stage_controller.py` puede ser reemplazado por otro controlador desarrollado.

---

## 2. Módulos auxiliares

Para el desarrollo de las actividades de este laboratorio se han desarrollado módulos auxiliares en el lenguaje **Python**<sup>4</sup> para facilitar y extender las funcionalidades básicas.

### 2.1. Programa de ejemplo

El archivo `scripts/stage_controller.py` corresponde a un programa de ejemplo en python. Se recomienda examinarlo cuidadosamente y utilizarlo como base para desarrollar las actividades del laboratorio.

### 2.2. Módulo de utilidades

El archivo `scripts/robot_utilities.py` contiene funciones auxiliares. Estas funciones se encuentran encapsuladas en la clase `Robot`

A continuación, se describen brevemente las más relevantes. Para mayor información, analizar el código fuente.

`nSteps(nL, nR)` Esta función le indica al robot mover la rueda izquierda `nL` pulsos de encoder y la derecha `nR` pulsos, respectivamente.

`show_distance()` Entrega la distancia recorrida.

`reset_perim()` Asigna 0.0 al contador de distancia recorrida.

`distances` No corresponde a una función sino a una variable de la clase `Robot` que contiene la última lectura del sensor de distancia. Es de tipo lista.

`laser_angles` Una variable tipo lista, que contiene los ángulos a los que corresponden cada una de las lecturas de `distances`.

---

<sup>4</sup><http://www.python.org>