

Práctica 1

Introducción a la Programación Robótica

Explorar y usar el paquete de ejemplo *Turtlesim*

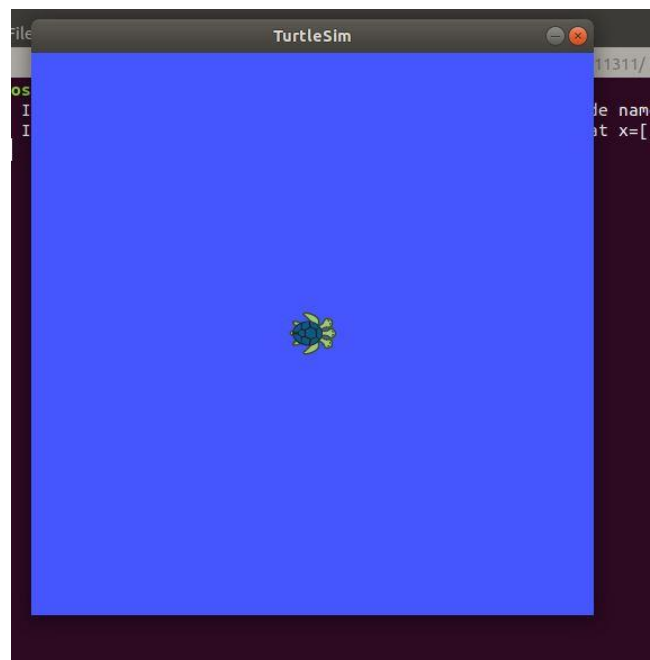
Enunciado

En esta primera práctica vamos a explorar el uso de un paquete de ejemplo como **Turtlesim**, aplicación viene preinstalada con ROS y que consiste en una simulación 2D de una tortuga. Turtlesim resulta útil para aprender los conceptos básicos de ROS como paso previo a trabajar con robots reales.

Aunque dedicaremos cierto tiempo a simular el comportamiento de lo que se conoce como “robots diferenciales” en próximas sesiones, en este punto nos limitaremos a usar Turtlesim simplemente para revisar algunos de los conceptos e ideas que han ido apareciendo en la introducción a ROS.

En concreto:

- navegación por el *workspace*
- comunicación y componentes
- entornos de visualización



Pasos a realizar:

1. Comencemos simplemente poniendo en marcha ROS. Para ello, cabe iniciar en una terminal el **roscore**.

```
$ roscore
```

2. Vamos a trabajar con el paquete de ejemplo de la Tortuga (turtlesim). Abre otra pestaña en el terminal activo e introduce los siguientes comandos:

```
$ roscd turtlesim  
$ rosrun turtlesim turtlesim_node
```

Habrás observado que el comando **rosrun** lanza la aplicación con el simulador de Turtlesim. De momento, no podemos hacer nada con la pantalla emergente.

```
ros@ros-VirtualBox:~$ rosrun turtlesim turtlesim_node  
[ INFO] [1571275021.039054334]: Starting turtlesim with node name /turtlesim  
[ INFO] [1571275021.047563210]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445], theta=[0.000000]
```

Remarcar que **rosrun** espera al menos dos argumentos: el nombre de la aplicación y el ejecutable. El nombre de la aplicación coincide con el contenido de una carpeta existente en el *workspace*. Sin embargo, ¿dónde guarda ROS los ejecutables? Búscalo en la estructura de directorios de ROS.

3. Echemos un vistazo ahora a la lista de nodos mediante el comando **rostopic**. Recuerda que un nodo en ROS es un componente con una función diferenciada en el esquema de comunicación de procesos.

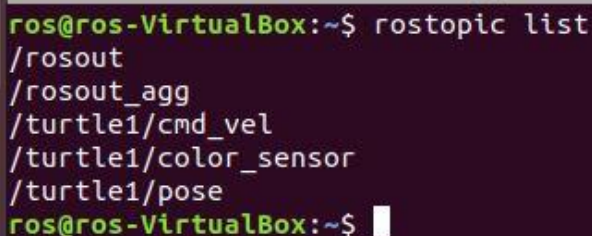
```
$ rostopic list
```

El comando devuelve la referencia al nodo *turtlesim*.

4. Hagamos ahora lo propio con la lista de **topics**. Recuerda que un *topic* en ROS es un bus (o canal) con nombre sobre el cual un nodo publica mensajes para que los reciban otros nodos.

Abre una nueva pestaña de terminal y escribe:

```
$ rostopic list
```

A screenshot of a terminal window with a dark background. The prompt is 'ros@ros-VirtualBox:~\$'. The command 'rostopic list' has been executed, and the output is displayed line by line: '/rosout', '/rosout_agg', '/turtle1/cmd_vel', '/turtle1/color_sensor', and '/turtle1/pose'. The prompt is now 'ros@ros-VirtualBox:~\$' with a cursor.

A través de los comandos anteriores, hemos podido averiguar que el paquete tiene:

- Un nodo en ejecución que se llama turtlesim
- Tres **topics** llamados **color_sensor**, **cmd_velocity** y **pose**

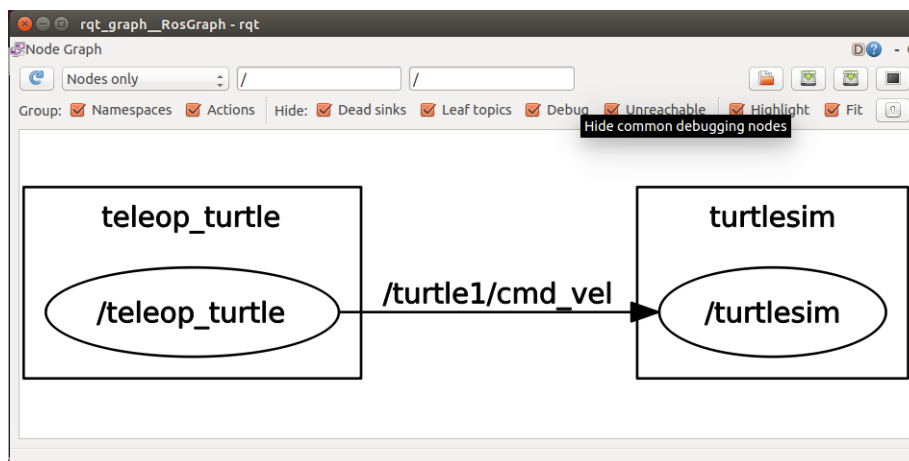
5. Ahora vamos a poner en ejecución otro nodo de ese paquete que nos permitirá usar el teclado (concretamente las teclas con flechas) para mover la tortuga, para lo cual debemos invocar a otra aplicación incluida en turtlesim.

```
$ rosrunc turtlesim turtle_teleop_key
```

En otra terminal comprueba mediante *rostopic* que se ha registrado el nuevo nodo que hemos lanzado (**teleop_turtle**).

6. Si usas las teclas con las flechas desde la terminal donde escribiste el comando anterior, deberías ver a la tortuga moviéndose por la pantalla. Cada vez que se presiona una tecla de flecha, el nodo *teleop_turtle* publica un mensaje en el *topic* "/turtle1/cmd_vel". El nodo *turtlesim* está suscrito a ese tema, recibe el mensaje y mueve la tortuga. Para ver gráficamente este proceso podemos utilizar la aplicación *graph* del paquete **rqt**:

```
$ rosrunc rqt_graph rqt_graph
```

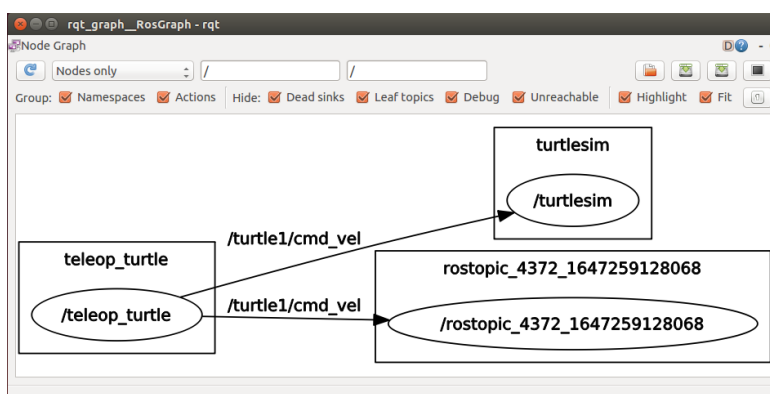


Vemos que el *topic* mediante el que se están comunicando es **cmd_velocity**. Además si en un nuevo terminal ejecutamos el comando **rostopic** con la opción "echo", obtendremos un eco por pantalla cada vez que se envíe este *topic*.

```
$ rostopic echo /turtle1/cmd_vel
```

7. Vamos a volver a mover la tortuga para comprobar que efectivamente el comando anterior muestra en pantalla información cada vez que se pulsa una flecha del teclado. Si volvemos a lanzar el gráfico, veremos que un nuevo nodo está suscrito al *topic* anterior, este nodo corresponde al comando *rostopic echo*, que se ha suscrito a dicho *topic*.

```
$ rosrun rqt_graph rqt_graph
```



8. Si volvemos a ver los *topics* de este paquete, pero con más detalle ...

```
$ rostopic list -v
```

... obtenemos algo así:

```
robot@alumnx-MIA:~/catkin_ws$ rostopic list -v

Published topics:
* /turtle1/color_sensor [turtlesim/Color] 1 publisher
* /turtle1/cmd_vel [geometry_msgs/Twist] 1 publisher
* /rosout [roscpp_msgs/Log] 3 publishers
* /rosout_agg [roscpp_msgs/Log] 1 publisher
* /turtle1/pose [turtlesim/Pose] 1 publisher

Subscribed topics:
* /turtle1/cmd_vel [geometry_msgs/Twist] 2 subscribers
* /rosout [roscpp_msgs/Log] 1 subscriber
```

Esta información nos viene a decir fundamentalmente que hay un nodo (sabemos que se llama *teleop_turtle*) que publica el topic *cmd_velocity* ("/turtle1" es el *namespace*). Este *topic* no es un tipo estándar (string, etc) sino que es del tipo *geometry/Twist*. Además hay un nodo que está suscrito al topic *cmd_velocity*, que ya sabemos que es *turtlesim*.

- Ya hemos visto que el mensaje enviado bajo el topic *cmd_vel* es de tipo *turtlesim/Velocity* pero lo podemos corroborar mediante la siguiente instrucción:

```
$ rostopic type /turtle1/cmd_vel
```

Además para ver la estructura interna de este tipo de mensaje disponemos del comando **rosmmsg**:

```
$ rosmmsg show geometry_msgs/Twist
```

Nos aparece que este mensaje está formado por otros dos de tipo básico (float32) que hacen referencia a la velocidad lineal y angular.

- Como sabemos el *topic* que envía el nodo *teleop_turtle* y también conocemos el formato del mensaje, vamos a probar a enviar manualmente desde terminal uno de estos mensajes (sin usar el nodo *teleop_turtle*) mediante el comando **rostopic**, cuya sintaxis es *rostopic pub [topic] [msg_type] [args]*

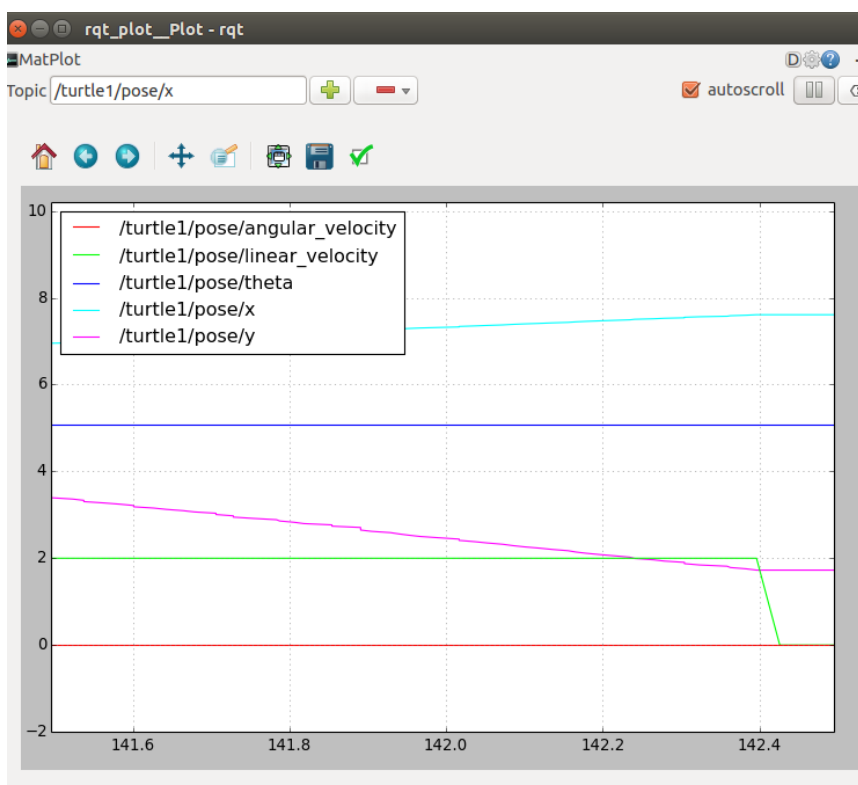
```
$ rostopic pub /turtle1/cmd_vel geometry_msgs/Twist <espacio en blanco + pulsar  
dos veces el tabulador > --once
```

Este comando ha enviado un mensaje con el *topic cmd_vel* que es de tipo *geometry/Twist* con valores de velocidad lineal y angular respectivamente. *Once* se usa para indicar que sólo se envíe este mensaje una única vez (no de manera periódica). Si queremos enviar varios topics de manera periódica entonces sustituimos *once* por *-r frecuencia_mensaje*.

11. Ros incorpora una interesante herramienta para visualizar la evolución de los mensajes enviados. Vamos a ver un ejemplo:

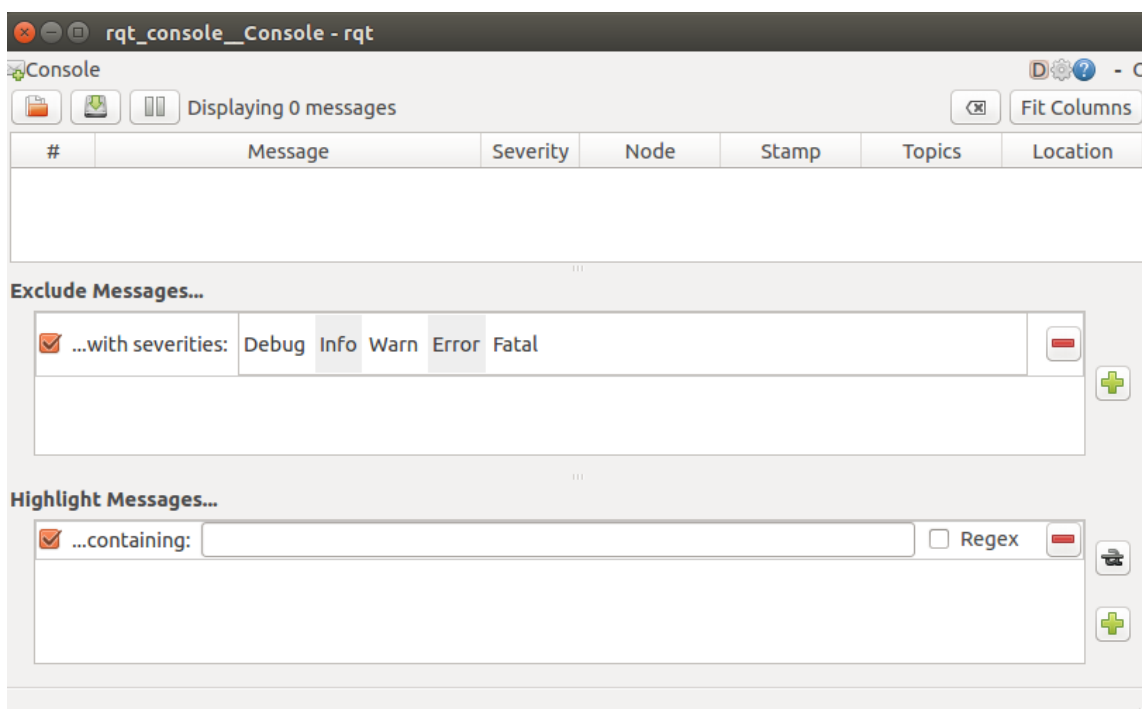
```
$ rosrun rqt_plot rqt_plot
```

En la ventana que aparece añadiremos el siguiente parámetro */turtle1/pose/x* y pulsamos el botón con el '+'. Añadimos otro más */turtle1/pose/y*. Podemos ver como evolucionan en el tiempo estos dos parámetros enviados mientras se mueve o movemos la tortuga.



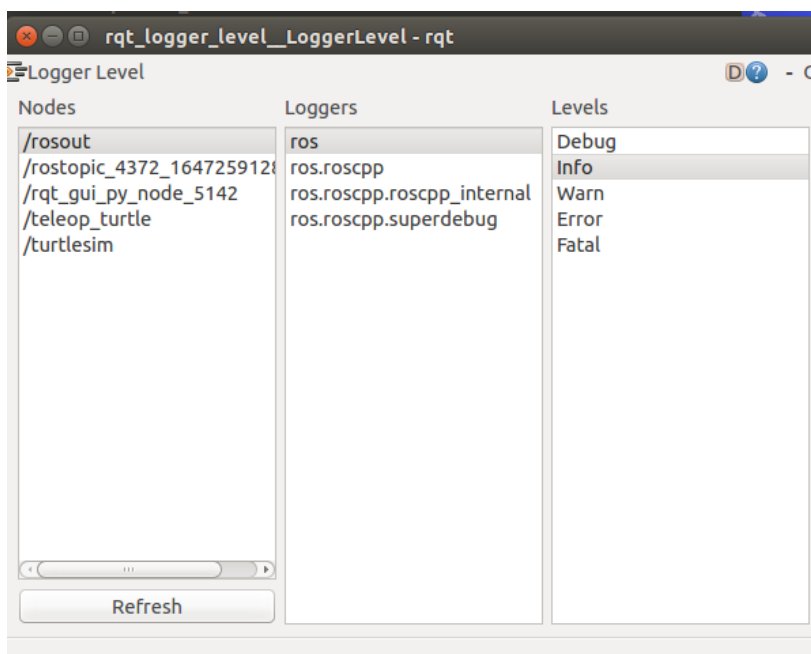
12. En ROS todos los mensajes que se muestran quedan almacenados en ficheros de log y que se pueden ver y filtrar de manera gráfica a través del siguiente comando:

```
$ rosrun rqt_console rqt_console
```



Para cambiar el nivel de “verbosity” ejecutamos:

```
$ rosrun rqt_logger_level rqt_logger_level
```



Podemos cambiar entre el nivel *Debug* de máxima verbosidad hasta el nivel de *Fatal* de mínima verbosidad. Si en el ejemplo anterior cambiamos a *Debug* vemos cómo nos

empiezan a aparecer de manera continuada los mensajes de posición de la tortuga. Se pueden establecer multitud de filtros a estos mensajes, bien sea por nodo, por *topic*, etc.

Revisión: explorando más características de Turtlesim

Acabaremos esta práctica explorando el paquete Turtlesim utilizando diferentes comandos de ROS para descubrir y modificar sus características.

Pasos a realizar:

1. Con el simulador Turtlesim y el nodo de teleoperación en funcionamiento, vamos a explorar los parámetros del sistema. Ejecuta los siguientes comandos y observa los resultados:

```
# rosparam list
```

Analiza la lista de parámetros disponibles. ¿Qué tipos de configuraciones se pueden modificar?

2. Vamos a experimentar modificando el color de fondo del simulador. Para ello:

```
# Primero, consultamos los valores actuales
# rosparam get /turtlesim/background_b
# rosparam get /turtlesim/background_g
# rosparam get /turtlesim/background_r

# Ahora, modificamos los valores (prueba diferentes números entre 0 y 255)
# rosparam set /turtlesim/background_r 150
# rosparam set /turtlesim/background_g 150
# rosparam set /turtlesim/background_b 150
```

- ¿Observas los cambios inmediatamente? ¿Qué necesitas hacer para que los cambios se apliquen?
 - Prueba diferentes combinaciones de colores.
3. Vamos a analizar la frecuencia con la que se publican los mensajes en algunos topics. Abre una nueva terminal y ejecuta:

```
# rostopic hz /turtle1/pose
```

- ☒ Mueve la tortuga usando las teclas de dirección mientras observas la salida del comando
- ☒ ¿Varía la frecuencia según el movimiento de la tortuga?

4. Utiliza **rqt_plot** para visualizar algunos parámetros de la tortuga:

```
# roslaunch rqt_plot rqt_plot
```

- ☒ Añade al gráfico el parámetro `/turtle1/pose/theta`
- ☒ En la misma gráfica, añade `/turtle1/cmd_vel/linear/x`
- ☒ Mueve la tortuga y observa cómo cambian las gráficas
- ☒ ¿Qué representan estos valores? ¿Cómo se relacionan con el movimiento de la tortuga?

5. Para finalizar, responde a las siguientes preguntas:

- ☒ ¿Qué información se muestra en el topic `/turtle1/pose`? Utiliza el comando adecuado para averiguarlo.
- ☒ Cuenta el número de nodos activos cuando tienes en ejecución el simulador y el teleop.
¿Cuántos hay? ¿Cuáles son?
- ☒ ¿Qué comando utilizarías para ver el tipo de mensaje que utiliza el topic `/turtle1/pose`?