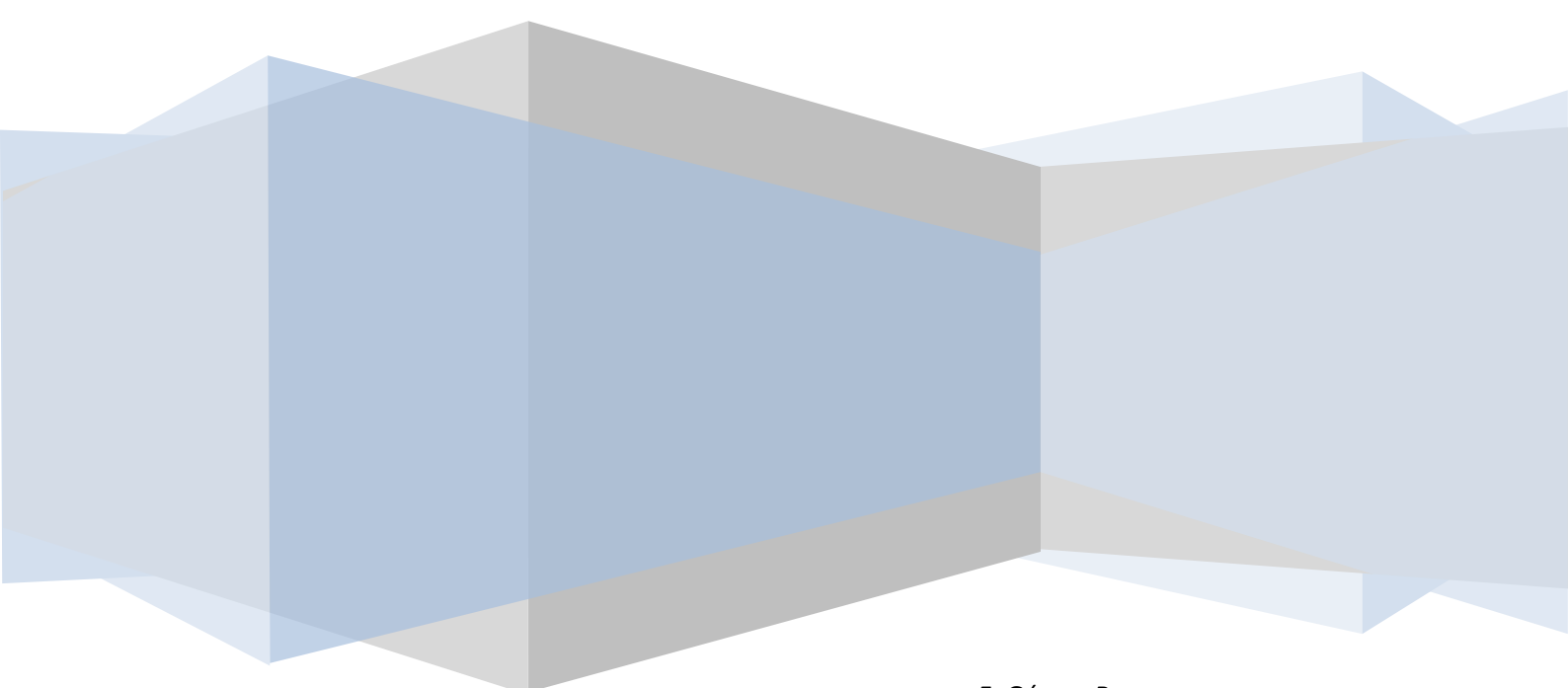


ROBÓTICA
CUARTO CURSO DEL GRADO EN
INGENIERÍA INFORMÁTICA



Práctica 3

Introducción al entorno de ROS.



F. Gómez Bravo
R. López de Ahumada
José Manuel Lozano

En esta práctica se describen y trabajan los conceptos básicos del sistema operativo ROS, las principales instrucciones básicas para ROS, así como la introducción a la programación de los robots móviles en el entorno de ROS

1. ¿Qué es ROS?

ROS son las siglas de Robot Operating System, en castellano sistema operativo robótico. ROS es un sistema operativo de código abierto que incluye conjunto de librerías y herramientas que ayudan a la hora de desarrollar aplicaciones o software para robots. Dicho de otro modo, ROS actúa como middleware entre un computador y el robot, incluyendo todo lo necesario para el desarrollo de la aplicación.

2. Opciones de instalación de ROS

ROS nació en el año 2007 y desde entonces ha estado evolucionando en diferentes distribuciones de forma conjunta con las distribuciones de Linux. Posteriormente, con el desarrollo de las versiones se detectó la necesidad de desarrollar nuevas funcionalidades debido a los retos que se enfrentan las nuevas tecnologías y se creó ROS 2. Las principales diferencias entre ROS 1 y ROS 2 radican en el sistema de comunicaciones implementado, ROS 1 hace uso de TCPROS que se basa en el uso de TCP/IP, mientras que ROS 2 se basa en Data Distribution Service (DDS) que está diseñado para aplicaciones en tiempo real. Además, ROS 1 tiene una estructura maestro-esclavo en el intercambio de los mensajes, lo cual es un sistema centralizado, mientras ROS 2 implementa un sistema de comunicaciones distribuido.

A todo ello, también es necesario destacar que tanto ROS 1 como ROS 2 tiene diferentes distribuciones, siendo las actualmente recomendadas Noetic Ninjemys para ROS 1 y Humble Hawksbill para ROS 2¹. Además de esto, es necesario tener en cuenta que las versiones instaladas en los robots del laboratorio son Indigo Igloo y Kinetic Kame solo disponibles en ROS 1.

Por último, cabe destacar que, para aprender a manejar ROS, es más sencillo comenzar por ROS 1 que por ROS 2, ya que los conceptos son más claros en estas distribuciones que en las siguientes.

3. Conceptos básicos de ROS

3.1. Sistema de ficheros:

- **Paquetes** (packages): son las principales unidades de organización del software en ROS. Estos paquetes pueden contener nodos, librerías dependientes de ROS, conjunto de datos o ficheros de configuración. Esta es la mínima unidad de software que se puede crear y lanzar.

¹ Documentación para cada versión: <https://docs.ros.org/>

- **Metapaquetes:** son paquetes que solo sirven para representar un grupo de otros paquetes relacionados.
- **Package Manifest:** los manifest o manifiestos (*nombrePaquete.xml*) proporcionan metadatos sobre un paquete. Esto incluye nombre, versión, descripción, dependencias, etc.
- **Tipo de mensaje o Message (msg):** definen las estructuras de datos para los mensajes enviados entre los nodos en ROS. Se almacenan en *paquete/msg/Message.msg*. Un mensaje es simplemente una estructura de datos, que comprende los campos con tipo. Se admiten los tipos primitivos estándar (integer, floats, booleanos, etc) , así como los arrays de tipos primitivos. Los mensajes pueden incluir estructuras y arrays anidados (similares a los *Structs* del lenguaje C).
- **Tipos de servicios Service (svr):** definen las peticiones y respuestas de las estructuras de los datos de los servicios en Ros. Se almacenan en *paquete/svr/Service.svr*.

3.2. Elementos de computación:

La comunicación y los procesos de datos en ROS se ejecutan en procesos Peer-To-Peer (P2P) o comunicación entre pares. Los conceptos mínimos son nodos (*Nodes*), maestro (*Master*), Servidor de Parámetros (*Parameter Server*), mensajes (*Message*), servicios (*Services*), temas (*Topics*) y bolsas (*Bags*). Todos ellos permiten ofrecer flujo de datos en diferentes direcciones.

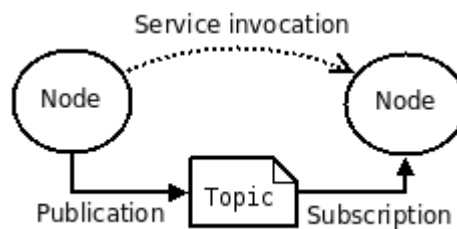
- **Nodos (Nodes):** los nodos son procesos que realizan cálculos. Ros está diseñado para ser modular, por ello un sistema robótico posee normalmente varios nodos. Cada nodo se encarga de una tarea concreta.
- **Maestro (Master):** El ROS Master proporciona el registro de nombres y búsqueda para el resto de los elementos de computación. Sin el maestro, los nodos no son capaces de encontrar o localizar a los demás nodos, ni pueden invocar servicios o intercambiar mensajes.
- **Servidor de parámetros (Parameter Server):** Forma parte del máster y permite que los datos sean almacenados en una ubicación central.
- **Mensajes (Messages):** elemento que utilizan los nodos para comunicarse entre ellos. Un mensaje tendrá una estructura de datos como se expuso en la sección 3.1 de esta guía.
- **Temas (Topics):** Mensajes que utilizan un sistema de transporte basado en la suscripción y publicación semántica. Un nodo envía un mensaje a través de una publicación en un *Tema o Topic*. Este mensaje llega a todos los nodos que estén suscritos a este *Tema o Topic*. En cada Tema o *Topic* puede haber múltiples suscriptores y múltiples editores o nodos emisores de información. Así mismo cada nodo puede estar suscrito o enviar información a diferentes Temas o *Topics*.
- **Servicios (Services):** El mecanismo de suscripción/publicación es un paradigma de comunicación muy flexible pero no es eficiente para comunicaciones basadas en peticiones y respuestas (*Request/Reponse*). Los servicios dan soporte a este tipo de comunicaciones. Para ello, un nodo ofrece un servicio con un nombre y otro nodo que actúa como cliente envía una petición y espera una respuesta.
- **Bolsas (Bags):** Las bolsas son un formato para guardar y reproducir datos de mensajes de ROS. Las bolsas son un mecanismo importante para el almacenamiento de datos, como los datos de los sensores, que pueden ser difíciles de recoger, pero es necesario para desarrollar y probar algoritmos.

3.3. Funcionamiento básico de ROS:

En ROS el Maestro (*Máster*) actúa como un servicio de nombres. El Maestro se encarga de almacenar/registrar los Temas o *Topics* y los servicios para que los Nodos hagan uso de ellos. Los Nodos se comunican con el Maestro para indicar la información que desean registrar. Así mismo, el Maestro ofrece información a los Nodos sobre otros nodos registrados y como se deben de hacer las conexiones adecuadamente. El Maestro también hará llamada a los Nodos cuando se produzcan cambios en la información de registro, lo que permite a los Nodos tener siempre la información actualizada y crear o deshacer las conexiones con otros nodos de manera dinámica.

Los Nodos se conectan a otros nodos directamente, mientras que el Maestro sólo proporciona información de búsqueda, al igual que un servidor DNS. Los Nodos que se subscriben a un tema solicitarán las conexiones a los nodos que publican sobre ese Tema o *Topic*, y se establecerá que la conexión a través de un acuerdo sobre el protocolo de conexión. El protocolo más común que se utiliza en un ROS se llama TCPROS, que utiliza sockets del protocolo TCP a través de una red IP.

En la siguiente imagen se puede observar gráficamente como sería la comunicación entre dos Nodos diferentes que publican información o se subscriben a un *Topic*, todo en la parte inferior de la imagen. En la parte superior vemos como se produce la comunicación de dos nodos mediante una petición/respuesta (Request/Response) de un servicio.

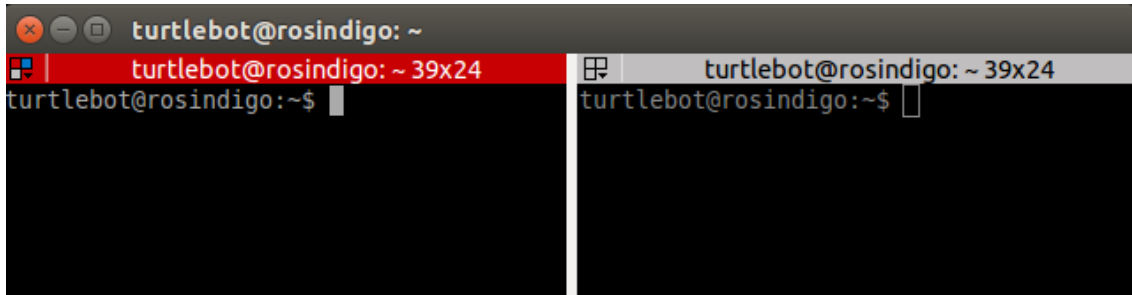


4. Configuración de la máquina virtual

Con el fin de poder ejecutar correctamente ROS y evitar instalar versiones dispares, se ha configurado una máquina virtual en Oracle VM VirtualBox. La máquina virtual ha sido configurada con una memoria RAM de 4GB, un disco duro de 30 GB, una unidad óptica, 64 MB de video y dos tarjetas de red. La primera tarjeta de red está configurada como NAT para que la máquina virtual tenga acceso Internet a través de la red de la Universidad de Huelva o en el domicilio o cualquier lugar deseado por el alumno. Sin embargo, la segunda tarjeta de red está configurada como "Adaptador puente" para poder conectar la máquina virtual a la red inalámbrica local (WLAN o WIFI) del laboratorio. Además de toda esta configuración, la máquina tiene habilitada la configuración necesaria para poder compartir el portapapeles con el sistema operativo anfitrión (normalmente Windows), así como también se ha habilitado la capacidad para reconocer memorias USB y poder así almacenar o rescatar la información necesaria de este tipo de memorias.

La máquina virtual configurada se llama ROS Aulas, y tiene instalado el sistema operativo Ubuntu en su versión 14.04.2. También, la máquina ya tiene instalado el software necesario

para poder ejecutar las instrucciones de ROS, para ello cuenta con la versión Indigo de ROS, debido a que es la versión más actualizada que soportan los robots que podemos utilizar en prácticas. Además de ello, la máquina virtual cuenta con una adaptación de la consola que permite tener varias terminales o consolas abiertas sobre una misma ventana como se observa en la siguiente figura. Esto se hace mediante la utilidad Terminator.



4.1. Exportar e importar la máquina virtual

Para realizar la instalación de la máquina en un ordenador distinto del que está instalada, tan solo es necesaria exportar la máquina, guardarla en una memoria USB y luego importarla en el ordenador deseado. A continuación, se exponen los principales pasos para exportar e importar la máquina virtual.

Para exportar la máquina virtual es necesario acceder a la pestaña “Archivo” de VirtualBox y pulsar sobre “Exportar servicio Virtualizado ...” o bien pulsar Ctrl+E. Es importante recordar que para exportar una máquina virtual esta debe estar apagada.

Luego es necesario seleccionar la máquina virtual que se quiere exportar. En este caso ROS_Aulas.

Una vez seleccionada la máquina virtual que se desea exportar, se debe seleccionar la carpeta de destino del fichero que contiene la máquina virtual exportada. Se recomienda exportar la máquina en primer lugar a un directorio del PC y luego pasarlo a una memoria USB para poder así acelerar el proceso de exportación. Además de esto, se debe seleccionar el formato del fichero de exportación, se recomienda dejar por defecto el formato “Open Virtualization Format 1.0”.

Para finalizar con la exportación, el software muestra un resumen de la máquina a exportar y solo queda por hacer clic sobre el botón “Exportar”. La exportación de la máquina puede tardar unos minutos. Una vez finalizada la exportación, la máquina debe de transferirse a una memoria USB o a un servicio de alojamiento de ficheros en la nube (por ejemplo, Mega, Consigna de la UHU, OneDrive, Drive, etc.).

Luego para importar un servicio o una máquina virtual, tan solo es necesario hacer doble clic sobre el fichero exportado y se abrirá el asistente de importación de máquinas virtuales. Al abrirse el asistente de importación, se podrá modificar la configuración de las CPUs utilizadas y adaptarlas a las características propias de cada PC, así como también se podrá adaptar la memoria RAM a las características de PC. Para realizar cualquier cambio tan solo es necesario hacer doble clic sobre el elemento que se desee modificar. Además, se puede modificar la carpeta base, es decir donde se almacenará la máquina virtual una vez importada, aunque esta

carpeta se puede modificar, se recomienda dejar la carpeta por defecto. Para finalizar el proceso tan solo es necesario pulsar “Importar” y esperar el tiempo necesario para importar la máquina virtual. Una vez importada, la máquina puede ser lanzada para trabajar con ella sin necesidad de realizar ninguna configuración más.

4.2. Credenciales de la máquina virtual

Para poder acceder a la máquina virtual será necesario conocer que el usuario de la máquina virtual es *turtlebot* y la contraseña del mismo es *turtlebot*. Además, este usuario tiene permisos de administrador para poder realizar cualquier modificación sobre la máquina virtual.

Resumen de credenciales:

- Usuario: turtlebot
- Contraseña o password: turtlebot

5. Primeros pasos en ROS

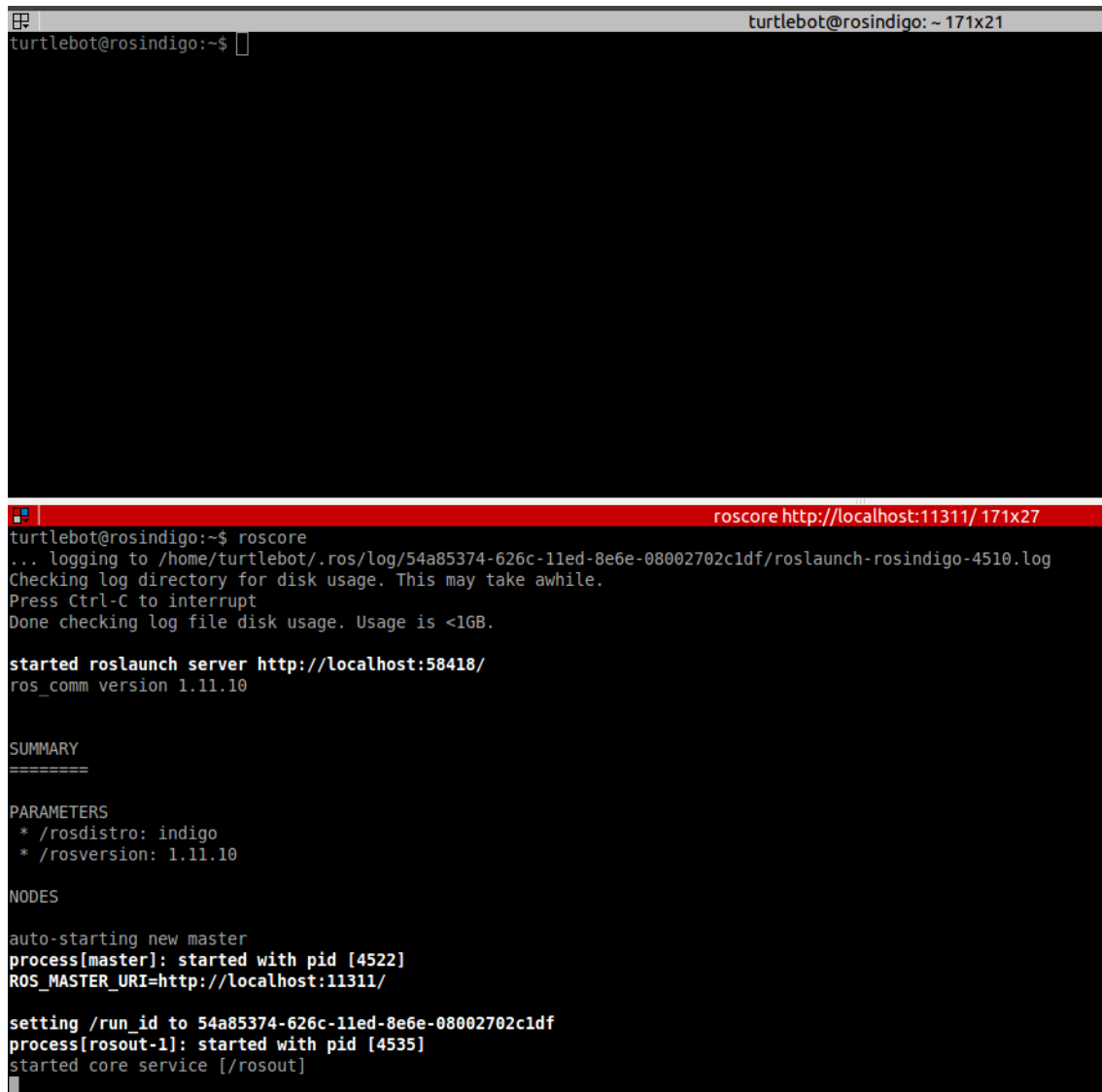
En estos primeros pasos en ROS se establecerán los primeros requisitos necesarios para poder trabajar con ROS de manera adecuada y ordenada. A lo largo de estos primeros pasos se establecerán los comandos básicos para comprender el funcionamiento de ROS y sus comandos o instrucciones asociados.

5.1. Instrucción *roscore*

En primer lugar, para poder utilizar ROS es necesario lanzar el nodo de ROS que actúa como Maestro o Máster de la red de nodos. Esto se realiza mediante la instrucción *roscore*, esta instrucción lanza al Maestro de la red, el servidor de parámetros de ROS y la consola de salida del sistema ROS que es un nodo denominado */rosout*. Es muy importante lanzar la instrucción *roscore* ya que, si no se lanza el Maestro, ROS no tendrá ninguna funcionalidad y no se podrá operar o trabajar correctamente.

Antes de lanzar la instrucción *roscore*, se procederá a abrir una terminal y pulsar botón derecho y crear al menos una división de la terminal para poder trabajar con *roscore* y otra con las instrucciones que se expondrán más adelante. Al lanzar *roscore* en una de las dos terminales abiertas sobre la misma ventana ofrece en una de las terminales la siguiente salida:

```
turtlebot@rosindigo:~$ roscore
```



```
turtlebot@rosindigo:~$ roscore
... logging to /home/turtlebot/.ros/log/54a85374-626c-11ed-8e6e-08002702c1df/roslaunch-rosindigo-4510.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://localhost:58418/
ros_comm version 1.11.10

SUMMARY
=====

PARAMETERS
* /rostdistro: indigo
* /rosversion: 1.11.10

NODES

auto-starting new master
process[roscout-1]: started with pid [4535]
ROS_MASTER_URI=http://localhost:11311/

setting /run_id to 54a85374-626c-11ed-8e6e-08002702c1df
process[roscout-1]: started with pid [4535]
started core service [/roscout]
```

Resultado 1: lanzamiento de roscore

La correcta ejecución de la instrucción devolverá al finalizar las palabras clave “*started core service [/roscout]*”. Si se observa el resto de la salida por consola de la orden *roscore*, se puede observar que el Maestro ha sido lanzado en el schoket localhost:11311, dicho de otra forma, se ha lanzado en una dirección de bucle local (IP 127.0.0.1) y en el puerto 11311. Además de ello, se observa que el Maestro se ejecuta en el proceso con identificador (PID) 4522, así como que la distribución utilizada de ROS es Indigo y la versión de ROS utilizada es la 1.11.10.

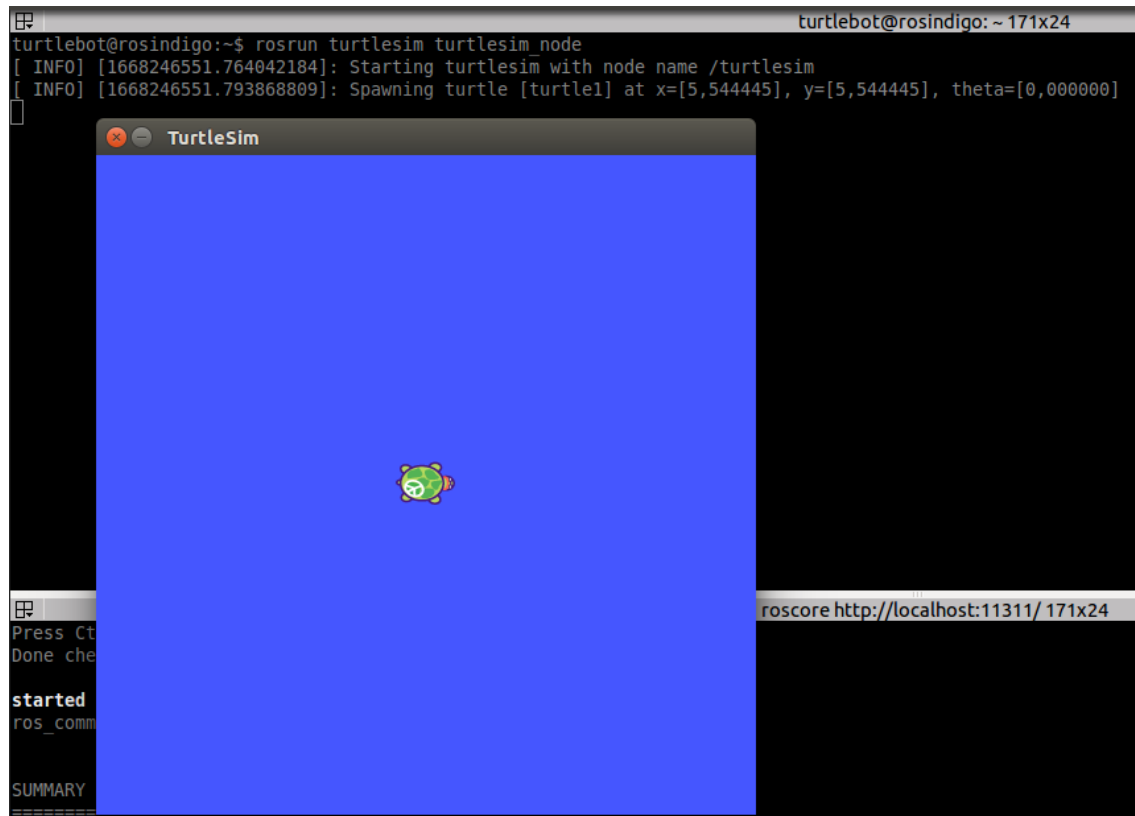
Nota: para detener cualquier la ejecución del Maestro o de cualquier nodo lanzado por consola se debe pulsar ctrl+C.

5.2. Instrucción *roslaunch*

La instrucción *roslaunch* es la instrucción utilizada para lanzar la ejecución de un nuevo nodo en el entorno ROS. En la presente guía se comenzará con el lanzamiento del lanzamiento de un simulador de robot para poder adquirir los conceptos asociados al funcionamiento del entorno

de ROS. La instrucción *roslaunch* puede lanzarse del siguiente modo *roslaunch nombre_del_paquete nombre_del_nodo* de este modo, se lanza el nodo con *nombre_del_nodo* que se encuentra ubicado en el paquete *nombre_del_paquete*. Llegados a este punto es necesario indicar que un nodo no es más que un programa escrito en C++ o Python con una funcionalidad concreta. En este caso, la funcionalidad del nodo a utilizar es representar a una tortuga en una ventana y simular el movimiento de un robot (la tortuga representada) en los ejes X, Y y Z. Para lanzar este nodo es necesario utilizar la instrucción *roslaunch turtlesim turtlesim_node*.

```
turtlebot@rosindigo:~$ roslaunch turtlesim turtlesim_node
```

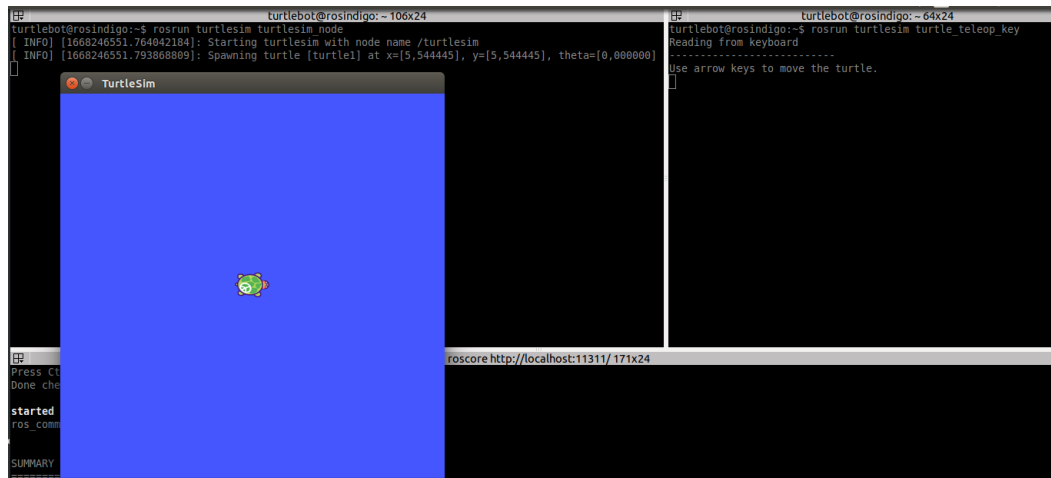


Resultado 2: lanzamiento de *roslaunch turtlesim turtlesim_node*

Al lanzar el comando *roslaunch turtlesim turtlesim_node* se debe lanzar una ventana con fondo azul y una tortuga que obedecerá a los mensajes enviados por otros nodos que desplegaremos más adelante. Es importante destacar que la tortuga no tiene porque ser la misma en todas las simulaciones, ya que la selección de la misma viene definida por una variable aleatoria.

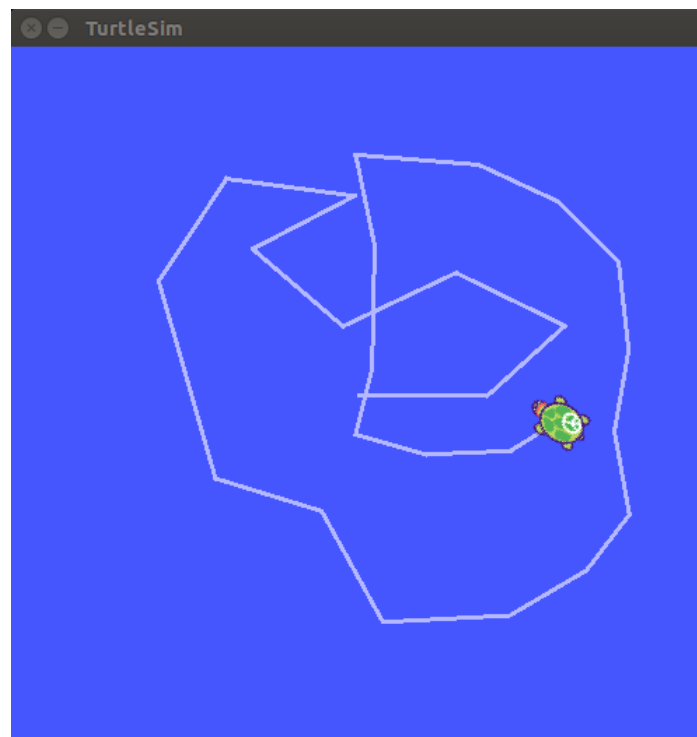
Una vez lanzado el nodo simulador de *turtlesim_node*, se procederá a lanzar un nuevo nodo que actuará como controlador de la tortuga mediante la pulsación de las flechas del teclado del PC. Para ello se hace uso de la instrucción *roslaunch* de nuevo, siendo en este caso el nombre del nodo a lanzar *turtle_teleop_key* y se encuentra ubicado en el paquete *turtlesim*. De este modo la instrucción a lanzar es *roslaunch turtlesim turtle_teleop_key*. Se recomienda dividir la terminal de nuevo para tener acceso a una nueva terminal sin necesidad de abrir una nueva y luego ejecutar el comando, ya que si no detendríamos al otro nodo o al Maestro.


```
turtlebot@rosindigo:~$ rosrn turtlesim turtle_teleop_key
```



Resultado 3: lanzamiento de `roslaunch turtlesim turtle_teleop_key`

Si se pulsa las flechas del teclado, la tortuga comenzará a moverse y dejará una estela blanca por donde ha ido pasando como se puede ver en la siguiente figura de resultado. La tecla arriba (↑) le indica a la tortuga que debe avanzar, la flecha derecha (→) le indica a la tortuga rotar en sentido horario (↻), la flecha izquierda (←) le indica a la tortuga rotar en sentido antihorario (↺) y la flecha abajo (↓) indica a la tortuga que debe retroceder o caminar hacia atrás. Gracias a la combinación de teclas se consigue como resultado el expuesto en la siguiente figura.



Resultado 4: lanzamiento de `roslaunch turtlesim turtle_teleop_key`

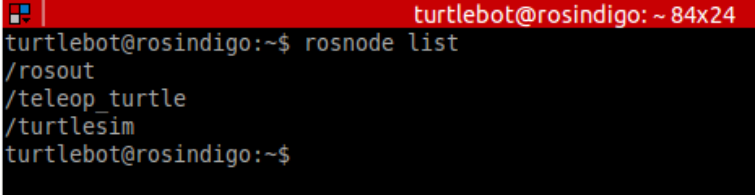
5.3. Instrucción *rostopic*

La instrucción *rostopic* permite conocer que nodos están en ejecución actualmente en el entorno ROS y la información asociada a estos. La instrucción *rostopic* está sobrecargada y en función del parámetro utilizado se podrá observar una información u otra. Las opciones de *rostopic* son las siguientes:

- *rostopic list*: muestra una lista con todos los nodos activos y registrados en el Maestro
- *rostopic info*: muestra información detallada en relación a un nodo concreto
- *rostopic ping*: permite conocer si el nodo sigue activo o no en el entorno ROS
- *rostopic cleanup*: permite eliminar los nodos huérfanos o sin conexión del entorno ROS
- *rostopic kill*: permiten terminar un nodo que se encuentra en ejecución
- *rostopic machine*: permite consultar los nodos que se están ejecutando en un PC concreto o una lista de PCs, ya que ROS puede trabajar con varios PCs a la vez

Los comandos más útiles para esta guía son *rostopic list* y *rostopic info*. Un ejemplo de *rostopic list* se encuentra en la siguiente figura. En ella se observa que actualmente en el entorno ros existen tres nodos activos, siendo estos nodos */rostopic*, */teleop_turtle* y */turtlesim*. Los dos últimos nodos son los que se han lanzado en el punto anterior, el primero de ellos para controlar la tortuga y el segundo para simular la tortuga, es necesario destacar que los nodos en este comando aparecen ordenados en orden alfabético y no por orden de lanzamiento de los mismos.

```
turtlebot@rosindigo:~$ rostopic list
```



```
turtlebot@rosindigo:~$ rostopic list
/rostopic
/teleop_turtle
/turtlesim
turtlebot@rosindigo:~$
```

Resultado 5: lanzamiento de *rostopic list*

Si se lanza el comando *rostopic info nombre_del_nodo*, permite conocer información de manera detallada del nodo que se pase por parámetro mediante la variable *nombre_del_nodo*. Un ejemplo de la misma para el nodo */turtlesim* se muestra a continuación. Lo recomendable es consultar primero la lista de nodos y luego la información de cada uno de los nodos.

```
turtlebot@rosindigo:~$ rostopic info /turtlesim
```

```
turtlebot@rosindigo:~$ rosnod info /turtlesim
-----
Node [/turtlesim]
Publications:
 * /turtle1/color_sensor [turtlesim/Color]
 * /rosout [rosgaph_msgs/Log]
 * /turtle1/pose [turtlesim/Pose]

Subscriptions:
 * /turtle1/cmd_vel [geometry_msgs/Twist]

Services:
 * /turtle1/teleport_absolute
 * /turtlesim/get_loggers
 * /turtlesim/set_logger_level
 * /reset
 * /spawn
 * /clear
 * /turtle1/set_pen
 * /turtle1/teleport_relative
 * /kill

contacting node http://localhost:52665/ ...
Pid: 5350
Connections:
 * topic: /rosout
   * to: /rosout
   * direction: outbound
   * transport: TCPROS
 * topic: /turtle1/cmd_vel
   * to: /teleop_turtle (http://localhost:60819/)
   * direction: inbound
   * transport: TCPROS

turtlebot@rosindigo:~$
```

Resultado 6: lanzamiento de `roslaunch turtlesim turtlesim.launch`

Como se puede observar en la figura anterior, la primera información que aparece es el nombre del nodo `"/turtlesim"`, luego aparecen los topics sobre los que el nodo `/turtlesim` puede escribir o publicar. Posteriormente, se muestra el topic al que está suscrito o escucha el nodo `/turtlesim`, así como los servicios que posee el nodo `/turtlesim`. Por último, aparece el puerto TCP que está usando el nodo, su PID y las conexiones que tiene activas ahora mismo el nodo que son con `/rosout` a través del topic `/rosout` y `/teleop_turtle` a través del topic `/turtle1/cmd_vel`, así como también se puede observar si la conexión es entrante (recibe información y mensajes) o saliente (envía información y mensajes).

5.4. Instrucción `rostopic`

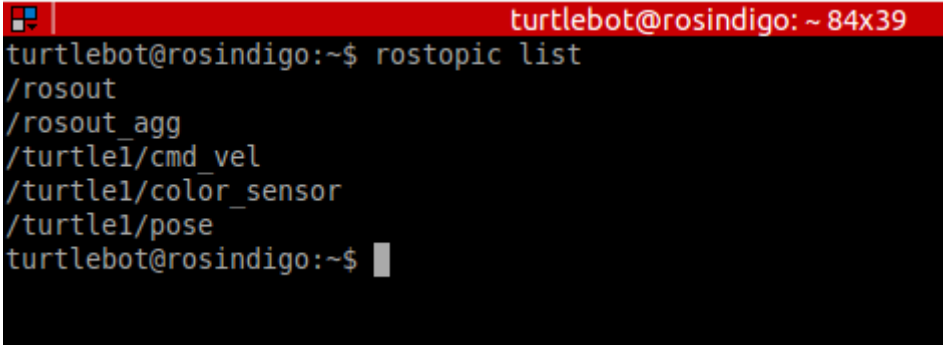
Otra instrucción muy interesante es la instrucción `rostopic`, la cual permite visualizar información en relación a los topics usados por los nodos que se encuentran lanzados, es decir, `rostopic` permite observar cuáles son los canales establecidos para el intercambio de mensajes o información entre nodos activos. Esta instrucción al igual que la anterior también un conjunto de opciones, las cuales son:

- `rostopic list`: muestra una lista con los topics activos en el entorno ROS
- `rostopic info`: muestra información completa acerca del topic pasado por parámetro
- `rostopic type`: identifica el tipo de mensaje usado en el topic pasado por parámetro
- `rostopic echo`: muestra por consola los mensajes intercambiados en el topic pasado por parámetro
- `rostopic pub`: permite publicar en el topic pasado por parámetro la información pasada como parámetro

- *rostopic find*: busca los topics activos en función del tipo pasado como parámetro
- *rostopic hz*: muestra la frecuencia con la que se publica en el topic pasado por parámetro
- *rostopic bw*: muestra el ancho de banda utilizado por el topic pasado por parámetro

Las instrucciones más útiles para esta guía son *rostopic list*, *rostopic info*, *rostopic type*, *rostopic echo* y *rostopic pub*. Un ejemplo de *rostopic list* se encuentra en la siguiente figura. En ella se observa los topics */rosout*, */rosout_agg*, */turtle1/cmd_vel*, */turtle1/color_sensor* y */turtle1/pose*. Los dos primeros topics están relacionados con el nodo *rosout*, mientras que los tres últimos están relacionados con los nodos */turtlesim* y */teleop_turtle*. El topic */turtle1/cmd_vel* se utiliza para indicar a la tortuga simulada a que velocidad debe moverse y */turtle1/pose* es utilizado por la tortuga simulada para indicar su posición, su rotación, y su velocidad (lineal y angular).

```
turtlebot@rosindigo:~$ rostopic list
```



```
turtlebot@rosindigo:~$ rostopic list
/rosout
/rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
turtlebot@rosindigo:~$
```

Resultado 7: lanzamiento de *rostopic list*

Si se desea consultar la información de un topic por ejemplo */turtle1/pose* tan solo es necesario indicar la siguiente instrucción *rostopic info nombre_del_topic* para el caso concreto *rostopic info /turtle1/pose*. El resultado ofrecido es tipo de mensajes utilizados por el topic, en este caso */turtlesim/Pose*, si se ejecutará la instrucción *rostopic type /turtle1/pose* se obtendría el mismo tipo de mensaje que el aquí mostrado. Además de esta información, la instrucción también ofrece conocer los Publishers o publicadores de información asociados a este topic, en este caso el nodo */turtlesim* y los nodos subscriptores o receptores de la información, actualmente no existe ninguno.

```
turtlebot@rosindigo:~$ rostopic info /turtle1/pose
```

```
turtlebot@rosindigo:~$ rostopic info /turtle1/pose
Type: turtlesim/Pose

Publishers:
 * /turtlesim (http://localhost:52665/)

Subscribers: None

turtlebot@rosindigo:~$
```

Resultado 8: lanzamiento de *rostopic info /turtle1/pose*

Mediante la instrucción *rostopic echo nombre_del_topic*, se pueden obtener los mensajes que se intercambian con ese topic en tiempo real. Por ejemplo, si se utiliza la instrucción *rostopic echo /turtle1/pose*, se conoce en cada instante la posición X e Y de la tortuga, la rotación de esta mediante la variable theta, la velocidad lineal y la velocidad angular. Si la tortuga está detenida, la información se actualiza, pero no cambia, mientras que si la tortuga está en movimiento se observa como la información además de actualizarse, se modifica. En la siguiente captura se puede observar los valores de la tortuga en reposo porque no varían ni la velocidad angular ni lineal.

Se recomienda probar a mover la tortuga con el teclado y observar cómo cambia la información del topic */turtle1/pose*.

```
turtlebot@rosindigo:~$ rostopic echo /turtle1/pose
```

```
turtlebot@rosindigo: ~ 84x
^Cturtlebot@rosindigo:~$ rostopic echo /turtle1/pose
x: 5.544444561
y: 5.544444561
theta: 0.0
linear_velocity: 0.0
angular_velocity: 0.0
---
x: 5.544444561
y: 5.544444561
theta: 0.0
linear_velocity: 0.0
angular_velocity: 0.0
---
x: 5.544444561
y: 5.544444561
theta: 0.0
linear_velocity: 0.0
angular_velocity: 0.0
---
```

Resultado 9: lanzamiento de *rostopic echo /turtle1/pose*

La instrucción *rostopic pub* será descrita más adelante en esta guía, ya que será utilizada para una demostración.

5.5. Instrucción *rosmmsg*

La instrucción *rosmmsg* permite consultar la información relacionada con los mensajes creados por algún paquete del entorno de ROS. Este comando también acepta opciones las cuales son las siguientes:

- *rosmmsg list*: lista todos los mensajes creados en el entorno de ROS, aunque no estén en uso actualmente
- *rosmmsg show*: muestra la estructura del mensaje pasado por parámetro
- *rosmmsg package*: muestra todos los mensajes creados en un paquete. Normalmente los paquetes utilizan más mensajes que los que se crean en ellos mismos, es decir, normalmente un mensaje puede utilizar mensajes creados en otros paquetes.
- *Rosmsgs packages*: muestra todos los paquetes que contienen mensajes

La instrucción más útil de todas ellas es *rosmmsg show nombre_del_mensaje*, ya que esta instrucción permite conocer la información que contiene el mensaje. Esta instrucción ofrece el nombre de las variables del mensaje y los tipos de las mismas. Por ejemplo, el mensaje utilizado por */turtle1/cmd_vel* es *geometry_msgs/Twist* pudiendo ser este obtenido mediante la instrucción *rostopic type /turtle1/cmd_vel*. Si se ejecuta la instrucción *rosmmsg show geometry_msgs/Twist* se obtiene que las variables que incluye el mensaje son dos vectores de tres posiciones, siendo el primer vector destinado a la velocidad lineal (velocidad lineal eje X, Y y Z) y el segundo de ellos destinado a almacenar la velocidad angular (velocidad angular eje X, Y y Z). Además, se observa que todas estas variables son float de 64 bits.

```
turtlebot@rosindigo:~$ rosmmsg show geometry_msgs/Twist
```

```
turtlebot@rosindigo:~$ rosmmsg show geometry_msgs/Twist
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z
```

Resultado 10: lanzamiento de *rosmmsg show geometry_msgs/Twist*

Existe la posibilidad de consultar el tipo de mensaje utilizado en un topic haciendo uso de tuberías en Linux, la instrucción concreta es *rostopic type nombre_del_topic | rosmmsg show*. En el caso de querer consultar el tipo de mensaje para el topic */turtle1/cmd_vel*, se puede consultar mediante tuberías con la siguiente instrucción *rostopic type /turtle1/cmd_vel | rosmmsg show*.

```
turtlebot@rosindigo:~$ rostopic type /turtle1/cmd_vel | rosmmsg show
```

```
turtlebot@rosindigo:~$ rostopic type /turtle1/cmd_vel | rosmmsg show
geometry_msgs/Vector3 linear
float64 x
float64 y
float64 z
geometry_msgs/Vector3 angular
float64 x
float64 y
float64 z
turtlebot@rosindigo:~$
```

Resultado 11: consulta tipo de mensaje mediante tubería

5.6. Instrucción *rospack*

La instrucción *rospack* incluyen un conjunto de opciones que permiten trabajar con los paquetes del entorno ROS, es decir con cada conjunto de programas del entorno ROS. Las principales opciones de *rospack* son:

- *rospack list*: esta instrucción lista todos los paquetes del entorno ROS disponibles y donde está ubicado (ruta)
- *rospack find*: esta instrucción busca el paquete pasado por parámetro y si está presente en el entorno devuelve su ruta, en caso de no estar presente ofrece un error
- *rospack depends1*: indica las dependencias directas del paquete pasado como parámetro
- *rospack depend*: indica todas las dependencias del paquete pasado como parámetro

Un ejemplo de *rospack find* se encuentra a continuación, se ha utilizado para buscar el paquete de *turtlesim* y conocer su ruta. Así mismo en la figura siguiente se observa también el error devuelto en el caso de buscar un paquete que no existe en el entorno.

```
turtlebot@rosindigo:~$ rospack find turtlesim
```

```
turtlebot@rosindigo:~$ rospack find turtlesim2
```

```
turtlebot@rosindigo:~$ rospack find turtlesim
/opt/ros/indigo/share/turtlesim
turtlebot@rosindigo:~$ rospack find turtlesim2
[rospack] Error: package 'turtlesim2' not found
turtlebot@rosindigo:~$
```

Resultado 12: ejemplos de *rospack find*

La instrucción *rospack depends1* muestra una lista de las dependencias directas del paquete pasado como parámetro, esto es útil para comprobar si todos los paquetes necesarios están instalados o falta alguna dependencia. Es posible que sea necesario hacer uso antes de esta instrucción de la instrucción *rosdep init* y *rosdep update*, estas instrucciones inician y actualizan las listas de paquetes disponibles. En el caso actual, se puede comprobar las diferentes dependencias del paquete *turtlesim* en la siguiente figura.

```
turtlebot@rosindigo:~$ rospack depends1 turtlesim
```

```
turtlebot@rosindigo:~$ rospack depends1 turtlesim
geometry_msgs
message_runtime
roscconsole
roscpp
roscpp_serialization
roslib
rostime
std_msgs
std_srvs
turtlebot@rosindigo:~$
```

Resultado 13: ejemplos de rospack dependencias directas

La instrucción *rospack depends* muestra una lista de las dependencias del paquete pasado como parámetro, esto es útil para comprobar si todos los paquetes necesarios están instalados o falta alguna dependencia. En el caso actual, se puede comprobar las diferentes dependencias del paquete *turtlesim* en la siguiente figura.

```
turtlebot@rosindigo:~$ rospack depends turtlesim
```

```
turtlebot@rosindigo:~$ rospack depends turtlesim
cpp_common
rostime
roscpp_traits
roscpp_serialization
genmsg
genpy
message_runtime
std_msgs
geometry_msgs
catkin
gencpp
genlisp
message_generation
roscbuild
roscconsole
roscgraph_msgs
xmlrpcpp
roscpp
rospack
roslib
std_srvs
turtlebot@rosindigo:~$
```

Resultado 14: ejemplos de rospack para todas las dependencias

5.7. Instrucción *roscd*

La instrucción *roscd* permite al usuario cambiar la ruta de la consola directamente al directorio donde se encuentra el paquete pasado como parámetro. Dicho de otro modo, al usar *roscd* y el nombre de un paquete se cambia de directorio estableciendo como directorio activo aquel donde se ubica el paquete. Esto sería similar a la instrucción *cd* de Linux seguida de la ruta a la que se desea ir. Por ejemplo, si se usa *roscd turtlesim* el directorio activo cambia a */opt/ros/indigo/share/turtlesim* como se puede observar en la siguiente figura. Una vez en el directorio se podrá listar los ficheros mediante *ls* o realizar cualquier otra operación de consola.


```
turtlebot@rosindigo:~$ roscd turtlesim/
```

```
turtlebot@rosindigo:~$ roscd turtlesim/  
turtlebot@rosindigo:/opt/ros/indigo/share/turtlesim$
```

Resultado 15: ejemplos de roscd

5.8. Demostración de escritura en un topic

En este momento se procede a retomar el uso de la instrucción *rostopic* con la opción *pub* con el fin de escribir o publicar en topic. En el caso de esta guía, la información se publicará en el topic */turtle1/cmd_vel*. Este topic permite indicarle a la tortuga simulada mediante un mensaje *geometry_msgs/Twist* una velocidad lineal y una velocidad angular mediante vectores de tres dimensiones como se determinó en el punto 5.5 de esta guía. Una vez se conoce la estructura del mensaje que se utiliza en el topic, se procederá a describir la instrucción necesaria para publicar en un topic. La instrucción es

```
rostopic pub -r 1 /nombre_del_topic tipo_del_mensaje 'parametro1' 'parametro2' ...  
'parametro n'.
```

Antes de comenzar, se recomienda detener la ejecución del nodo *turtlesim* y del nodo *turtle_teleop_key*, y posteriormente lanzar el nodo *turtlesim* de nuevo. Esta recomendación es para poder iniciar el movimiento de la tortuga desde el punto original.

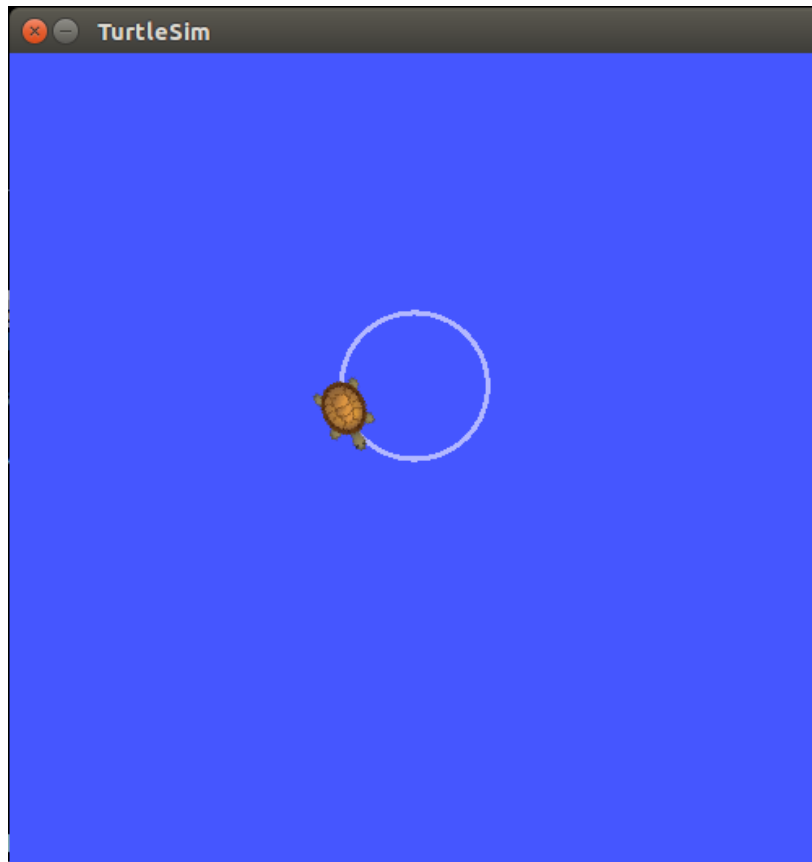
Para comenzar esta demostración, se debe lanzar un nuevo nodo *turtlesim* mediante *roslaunch turtlesim turtlesim* e iniciar una consola donde se pueda observar la pose de la tortuga mediante *rostopic echo /turtle1/pose*. Una vez realizado estos pasos, lanzar el comando *rostopic pub -r 1* del siguiente modo:

```
turtlebot@rosindigo:~$ rostopic pub -r 1 /turtle1/cmd_vel geometry_msgs/Twist '[2.0,0.0,0.0]' '[0.0,0.0,2]'
```

Donde los parámetros empleados son como velocidad lineal 2 m/s en el eje X y como velocidad angular 2 rad/s en el eje Z, de este modo la tortuga generará un círculo como se puede observar en la siguiente figura. Además, si se observa la terminal donde está lanzada la instrucción *rostopic echo /turtle1/pose*, se puede observar como la velocidad lineal y angular se mantienen constante pero la posición X, Y y theta están constantemente cambiando como se observa en la figura de Resultado 17.

Además de este modo de publicar en un topic de manera constante, también se puede publicar una sola vez en un topic, esto se consigue con la instrucción *rostopic pub -1 /nombre_del_topic tipo_del_mensaje 'parametro1' 'parametro2' ... 'parametro n'*. La principal diferencia es que en el caso anterior el parámetro era *-r* de recurrente cada 1 segundo, mientras que ahora el parámetro es *-1* que indica que solo se ejecuta una vez.

```
turtlebot@rosindigo:~$ rostopic pub -r 1 /turtle1/cmd_vel geometry_msgs/Twist '[2.0,0.0,0.0]' '[0.0,0.0,2]'
```



Resultado 16: simulación de la tortuga generando un círculo

```
turtlebot@rosindigo:~$ rostopic echo /turtle1/pose
```

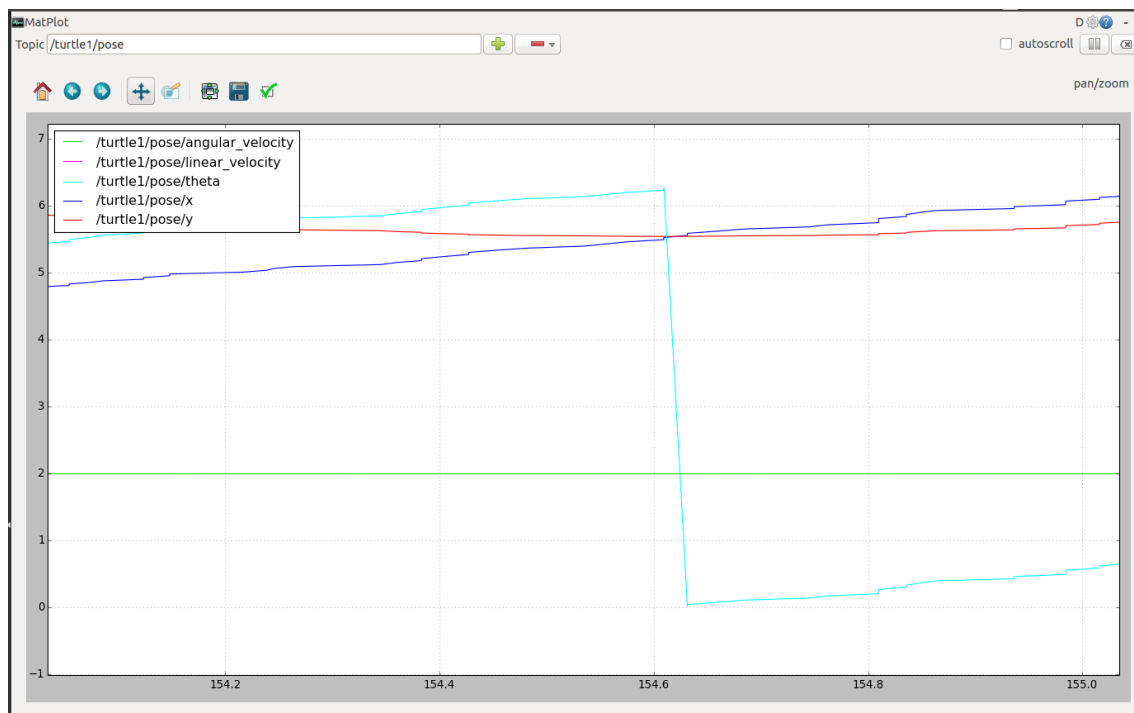
```
x: 6.52758264542
y: 6.50185537338
theta: 1.51228165627
linear_velocity: 2.0
angular_velocity: 2.0
---
x: 6.52843093872
y: 6.53384447098
theta: 1.54428160191
linear_velocity: 2.0
angular_velocity: 2.0
---
x: 6.52825546265
y: 6.56584358215
theta: 1.57628154755
linear_velocity: 2.0
angular_velocity: 2.0
---
```

Resultado 17: cambios en las variables X, Y y Theta en función de la posición de la tortuga

6. Visualización de variables de estado de la tortuga de forma gráfica

Para poder monitorizar el movimiento de la tortuga no solo se puede hacer uso de la consola mediante *rostopic echo*, sino que además se puede hacer uso de una herramienta gráfica que permite graficar los cambios realizados. La herramienta en cuestión se denomina *rqt_tplot* y permite dibujar o graficar los cambios de las variables numéricas de los topics. La herramienta no deja de ser un nodo subscritor de los topics y por tanto se debe lanzar como un nodo más. Para lanzar la herramienta *rqt_plot* se hace uso de la instrucción *roslaunch rqt_plot rqt_plot*, donde *rqt_plot* es tanto el nombre del nodo como el paquete donde está ubicado el nodo. Una vez lanzada la herramienta hay que indicar el topic que se desea escuchar o del que se desea recibir información. En el caso actual, el topic seleccionado debe ser */turtle1/pose* donde mostrará una gráfica similar a la siguiente.

```
turtlebot@rosindigo:~$ roslaunch rqt_plot rqt_plot
```



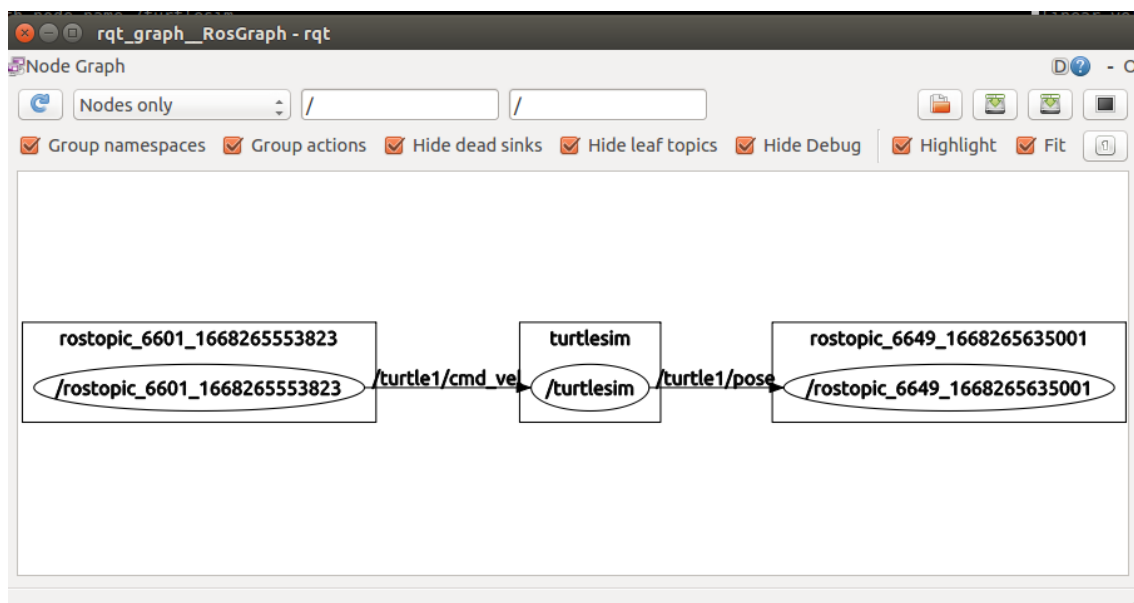
En la figura se observan en el eje X el tiempo y en el eje Y el valor de las variables, las variables velocidad linear y angular están en el punto 2, mientras que el resto de las variables aumentan o disminuyen. En especial hay que dedicar especial atención a la variable theta en celeste que cambia de un valor algo mayor que 6 puntos a 0 y luego vuelve a crecer. Esto se debe a que la orientación de la tortuga se mide en radianes, comienza en cero radianes y el máximo es 2π radianes.

7. Representación gráfica de la conexión entre nodos

La conexión o conexiones presente en el entorno de ROS entre los diferentes nodos se puede analizar mediante las instrucciones *rostopic* o *rostopic* y analizar los resultados de cada una de

las funciones para detectar las conexiones. ROS además cuenta con una herramienta gráfica que permite detectar estas conexiones de forma gráfica, la herramienta se denomina *rqt_graph*. A través de ella se pueden observar que nodos están interactuando entre sí y a través de que topics existe esa conexión. Un ejemplo de las conexiones y los nodos se observa en la siguiente figura. En la parte central está el nodo */turtlesim* que se lanzó anteriormente, que se encuentra suscrito (recibiendo información) del nodo */rostopic_6601_1668265553823* que es el nodo que se crea cuando la consola está escribiendo en el topic */turtle1/cmd_vel* donde se establece la velocidad con la que se quiere que gire la tortuga. A su vez, el nodo */turtlesim* esta actuando de Publisher o publicador (nodo emisor de información) para el nodo */rostopic_6649_1668265635001*, nodo creado cuando se hace uso de la instrucción *rostopic echo* y que está suscrito al topic */turtle1/pose*, siendo la conexión existente entre ambos nodos el topic */turtle1/pose*, donde se puede observar la posición X, Y, la rotación en Theta y la velocidad angular y lineal. La utilidad de la herramienta radica en determinar que nodo es el emisor de la información y cual el receptor sin necesidad de leer toda la información de cada uno de los nodos.

```
turtlebot@rosindigo:~$ rosrunc rqt_graph rqt_graph
```



8. Tareas

Para finalizar se le proponen dos tareas al alumnado para que prueben los conceptos explicados en la guía.

8.1. Consulta el tipo de mensaje del topic */turtle1/pose*

8.2. Consigue que la tortuga realice un círculo de Radio $R = (1, 2, 4, 8)$, con una velocidad inicial en el eje Y de 4 m/s.