



Informe de Práctica 4

Programación de Robots

David Ávila Jiménez || Pedro Antonio Aguilar Lima

Contenido

Resumen.....	3
Tutorial.....	3
1. Instalación y configuración de tu entorno ROS.	3
2. Creando un paquete ROS	3
3. Entendiendo los nodos ROS.....	4
4. Entendiendo topics de ROS.....	5
5. Publicación y suscripción de nodos en (C++)	6
Código de la práctica	6
Conclusión	8

Resumen

En este informe se recoge toda la información relacionada con la Práctica 4 sobre el entorno ROS. El objetivo de la práctica es aprender el funcionamiento del framework ROS, en donde se aprenderá los conceptos de como crear paquetes de ROS, suscribir y publicar topics.

Para ello se tendrá que realizar algunos de los tutoriales que vienen en la wiki sobre ROS, teniendo como objetivo final hacer uso del nodo turtlesim, una tortuga mostrada por pantalla que sigue un trazado marcado por el teclado, o por un .txt como es en este caso en la práctica y cuya posición debe ser mostrada todo el tiempo.

Tutorial

En este apartado comenzaremos hablando un poco del procedimiento seguido en el tutorial de la página. Los tutoriales para realizar para la práctica son los del apartado beginner level (si se quiere profundizar más se pueden realizar el resto):

1. Instalación y configuración de tu entorno ROS.

En este primer tutorial vamos a configurar nuestro espacio de trabajo. Así, realizaremos los primeros comandos en el directorio de nuestra carpeta personal, desde una terminal de Linux.

```
$ mkdir -p ~/Tortuga_ws/src
$ cd ~/Tortuga_ws/
$ catkin_make
```

Tras esto se habrá creado nuestro workspace Tortuga_ws, y se producirá la instalación de las carpetas build, devel y src, necesarias para la creación, compilación y ejecución de nuestra práctica. Por último, haremos los siguientes comandos:

```
$ source devel/setup.bash
$ echo $ROS_PACKAGE_PATH
```

El primer comando se deberá realizar siempre que se vaya a compilar paquetes, el segundo comando sirve para comprobar el path de nuestro entorno.

2. Creando un paquete ROS

Tras haber realizado la configuración del entorno ROS, el siguiente paso es crear nuestro primer paquete de ROS, para ello realizaremos los siguientes comandos en una terminal.

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg tortugo std_msgs roscpp geometry_msgs
```

El primer comando nos lleva al directorio src dentro de nuestro workspace y el segundo crea el paquete tortugo con las dependencias std_msgs, roscpp y geometry_msgs. Una vez hecho esto personalizamos el archivo package.xml que se encuentra dentro del paquete tortugo.

```

<?xml version="1.0"?>
<package>
  <name>tortugo</name>
  <version>0.1.0</version>
  <description>The tortugo package</description>

  <maintainer email="you@yourdomain.tld">Your Name</maintainer>
  <license>BSD</license>
  <url type="website">http://wiki.ros.org/beginner_tutorials</url>
  <author email="you@yourdomain.tld">Jane Doe</author>

  <buildtool_depend>catkin</buildtool_depend>

  <build_depend>roscpp</build_depend>
  <build_depend>geometry_msgs</build_depend>
  <build_depend>std_msgs</build_depend>

  <run_depend>roscpp</exec_depend>
  <run_depend>geometry_msgs</exec_depend>
  <run_depend>std_msgs</exec_depend>

</package>

```

Aquí podemos elegir una licencia, por defecto viene todos, pero nosotros la vamos a cambiar a BSD. También se puede cambiar el email autor, para que se puede contactar con la persona por alguien de la comunidad que necesite ayuda. Algo también interesante es que se pueden añadir nuevas dependencias aquí.

Tras realizar estos cambios volvemos al directorio raíz del workspace y ejecutamos la siguiente orden:

```

$ cd ~/catkin_ws
$ catkin_make

```

Al realizar catkin_make se compilará nuestro paquete con todas sus dependencias para que después pueda ser usado por un nodo.

3. Entendiendo los nodos ROS

Lo primero que debemos escribir en una nueva terminal es:

```
$ roscore
```

Esto lanzará el servidor de ros, apareciendo en nuestra terminal algo tal que así:

```

... logging to ~/.ros/log/9cf88ce4-b14d-11df-8a75-00251148e8cf/roslaunch-machine_name-1303
9.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://machine_name:33919/
ros_comm version 1.4.7

SUMMARY
=====

PARAMETERS
* /rosversion
* /rostdistro

NODES

auto-starting new master
process[master]: started with pid [13054]
ROS_MASTER_URI=http://machine_name:11311/

setting /run_id to 9cf88ce4-b14d-11df-8a75-00251148e8cf
process[roscout-1]: started with pid [13067]
started core service [/roscout]

```

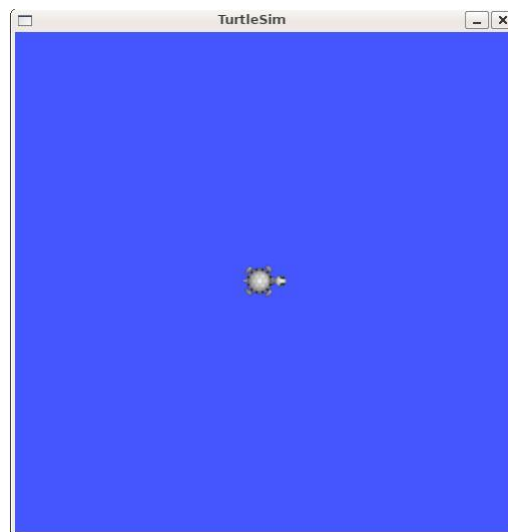
Tras esto si realizamos el siguiente comando en una nueva terminal:

```
$ rosnodet list
```

Se verán la lista de nodos activos, en este caso se verá **/rosout**. Ahora vamos a utilizar el siguiente comando:

```
$ rosrund turtlesim turtlesim_node
```

Esto hará que se lance el nodo turtlesim_node del paquete turtlesim y veremos que aparecerá una nueva ventana con una tortuga.



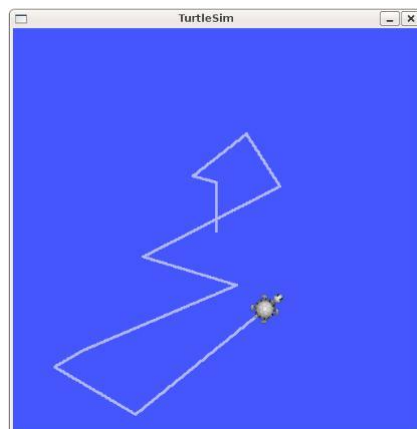
Si volvemos a realizar el comando **\$ rosnodet list**, aparecerá en la lista de nodos tanto **/rosout** como **/turtlesim**.

4. Entendiendo topics de ROS

En este apartado pasaremos simplemente mencionando algunos de los topics que se pueden utilizar desde la consola y que nos ofrecen funcionalidades nuevas de turtlesim.

```
$ rosrund turtlesim turtle_teleop_key
```

Lanza un nodo que se conecta a turtlesim y permite que se puedan mover la tortuga con las flechas del teclado.



```
$ rosrun rqt_graph rqt_graph
```

rqt_graph crea grafos de los nodos que tenemos activos, mostrando como están relacionados entre sí. El grafo que se muestra a continuación es un ejemplo de como estaría conectado todo en turtlesim una vez realizado rosrun a cada uno de los topics que se explican en este apartado.



Los siguientes topics que vamos a utilizar son:

```
$ rostopic hz /turtle1/pose
```

```
$ rosrun rqt_plot rqt_plot
```

El primer topic nos informa a la velocidad a la que se publican los datos, el segundo nos muestra una nueva ventana, con la que podremos ver gráfica con la posición que está siendo publicada de la tortuga.

5. Publicación y subscripción de nodos en (C++)

En nuestro paquete en dentro del directorio src podemos crear nodos publicadores y nodos suscriptores. En este apartado no explicaremos que hace cada código, ya que después se explicará el código de la práctica y ahí se ha realizado un nodo que es suscriptor y publicador al mismo tiempo.

Código de la práctica

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include "geometry_msgs/Twist.h"
#include "turtlesim/Pose.h"
#include "geometry_msgs/Vector3.h"
#include "csignal"
#include <sstream>
#include <fstream>
#include <cstdlib>
#include <iostream>
#include <cstring>
#include <string>
using namespace std;

void muestraCoordenadasCallback(const turtlesim::Pose& coordenadas)
{
    ROS_INFO_STREAM("Linear X: " << coordenadas.x << " Y: " << coordenadas.y << " theta: "
    << coordenadas.theta);
}
```

En primer lugar, añadimos las librerías necesarias para realizar la práctica (hay que decir que algunas no son utilizadas durante la práctica). Acto seguido encontramos la primera función que nos muestra las coordenadas de la tortuga que nos va llegando del topic al que nos hemos suscrito.

```

void signalHandler(int signum) {
    ROS_INFO_STREAM("CERRANDO NODO");
    ros::shutdown();
    exit(0);
}

```

La siguiente función que nos encontramos tiene como objetivo cerrar civilizadamente nuestro nodo creado. En estas dos funciones se usa `ROS_INFO_STREAM()`, en lugar de `ROS_INFO`, debido a que la primera función trabaja en C++ y la segunda en C y en esta práctica vamos a dedicarnos a C++.

```

int main(int argc, char **argv)
{
    char buffer[100];
    geometry_msgs::Twist datos_velocidad;
    ifstream fichero;

    ros::init(argc, argv, "practica");
    ros::NodeHandle n;
    ROS_INFO_STREAM("COMIENZA NODO");
    ros::Publisher pub = n.advertise<geometry_msgs::Twist>("turtle1/cmd_vel", 1000);
    ros::Subscriber sus = n.subscribe("turtle1/pose", 1000, muestraCoordenadasCallback);
    ros::Rate loop_rate(10);

    fichero.open("/home/viki/Tortuga_ws/src/tortugo/src/Datos.txt");

    if(!fichero){
        cout << "No se ha podido abrir el archivo. (" << endl;
        exit(EXIT_FAILURE);
    }
}

```

Aquí comienza el main del código y al comienzo lo que realizamos es declarar variables como puede `char buffer[100]`, que nos servirá para guardar los datos que vayamos leyendo del fichero. Iniciamos el nodo con `ros::init` y `ros::NodeHandle`, y avisamos al usuario de que se ha iniciado el nodo. El siguiente paso es crear un objeto publicador que se conectará al topic de la tortuga que controla su velocidad, para que le vayamos publicando los datos de velocidad linear y angular, leídos en nuestro fichero. Además crearemos el objeto suscriptor que se conectará al topic controla la posición de la tortuga, enviándonos datos que mostraremos con la función `muestraCoordenadasCallback`, ya explicada anteriormente. Por último, en este parte crearemos un tipo fichero y lo abriremos con la ruta donde se encuentra nuestro fichero, con los datos de la velocidad que deberá seguir la tortuga. En caso de no encontrarse se cerrará el nodo.

```

while (ros::ok() && fichero>>buffer){

    datos_velocidad.linear.x = atof(buffer);
    fichero >> datos_velocidad.linear.y
    >> datos_velocidad.linear.z >>
    datos_velocidad.angular.x >>
    datos_velocidad.angular.y;
    fichero >> buffer;
    datos_velocidad.angular.z = atof(buffer);
    pub.publish(datos_velocidad);
    ros::spinOnce();
    loop_rate.sleep();
}

fichero.close();
signalHandler(0);
ros::spin();
return 0;
}
fichero.close();
signalHandler(0);
ros::spin();
return 0;
}

```

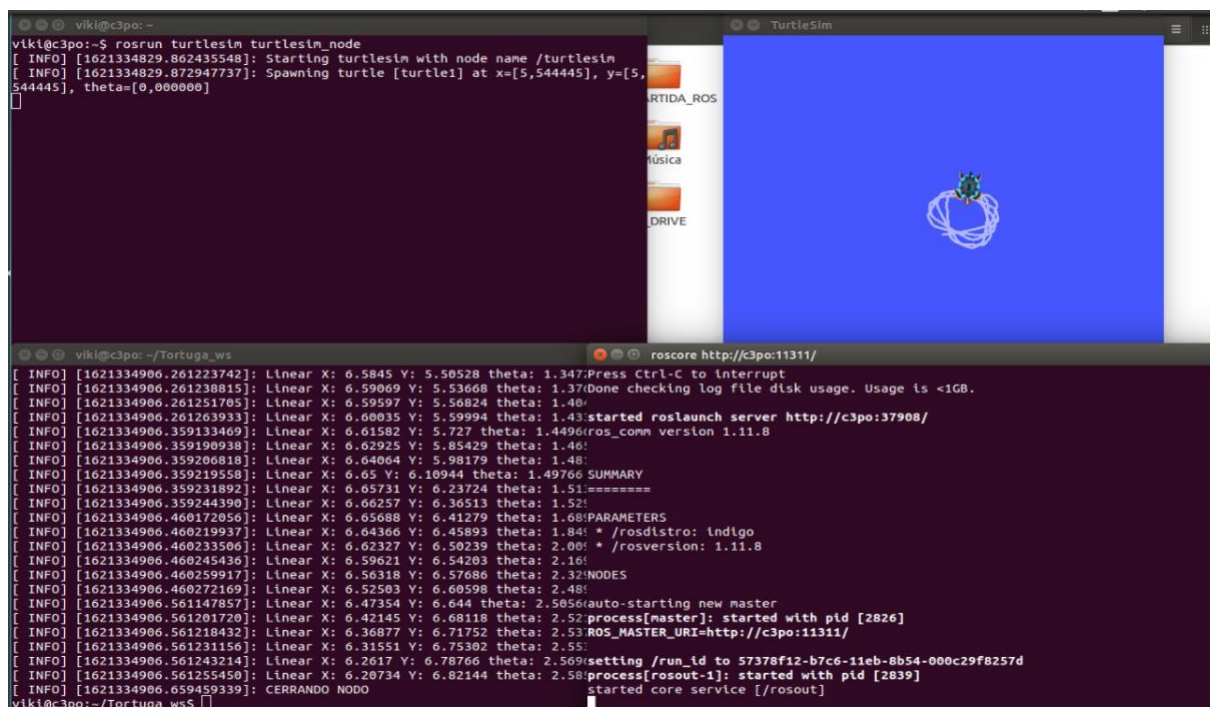

Ahora pasamos al bucle while que está controlado por `ros::ok()`, en caso de que se pulse Ctrl-C y por fichero `>> buffer` que comprobará si se ha llegado al final del fichero.

Dentro del bucle vamos guardando los diferentes datos de velocidad linear y angular que vamos leyendo, en el objeto `datos_velocidad`, y lo que hacemos acto seguido es publicarlos para que la tortuga pueda moverse y nos vaya enviando los datos de su posición.

Una vez fuera del bucle cerramos el fichero y llamamos a la función `signaHandler()`, para que nos cierre el nodo civilizadamente.

Conclusión

Tras realizar todos los tutoriales y realizar los objetivos de la práctica el resultado que quedaría en nuestro caso la tortuga sería el siguiente:



The screenshot displays a ROS environment. On the left, a terminal window titled 'viki@c3po: ~' shows the execution of `roslaunch turtlesim turtlesim_node`. It logs the start of the `turtlesim` node and the spawning of a turtle named `turtle1` at coordinates `x=[5.544445], y=[5.544445], theta=[0.000000]`. Below this, another terminal window titled 'viki@c3po: ~/Tortuga_ws' shows a stream of log messages from the `turtlesim` node, including linear and angular velocities (X, Y, theta) and a summary of parameters. On the right, a window titled 'TurtleSim' shows a blue square environment with a small green turtle icon moving in a circular path, leaving a trail of white lines.

En él, se puede observar cómo nuestro nodo creado se conecta a los topics correctamente y envía los datos a la tortuga para que ella realice los movimientos necesarios, enviando en todo momento su posición. Cuando la tortuga deja de recibir datos, esta se para y nuestro nodo deja de recibir datos ya que se para.