Escuela Politécnica Superior
Grado en Ingeniería Robótica
Programación de Robots

Universitat d'Alacant
Universidad de Alicante

Ejercicio Práctico 2: Primeros pasos en ROS

Profesor: Antonio Requena Jiménez

Instrucciones

A lo largo de esta práctica vamos a aprender a crear un espacio de trabajo ROS, crear paquetes ROS y crear mensajes para que otros nodos publiquen ese tipo de mensajes. Además, aprenderemos a lanzar varios nodos desde un fichero *.launch.

1. Creación del espacio de trabajo ROS

Para crear un espacio de trabajo ROS lo único que tenemos que hacer es navegar hasta el directorio en el que queramos crear el espacio de trabajo y ejecutar los comando que aparecen a continuación. Se recomienda crearlo en la carpeta raíz de Ubuntu y con el nombre *catkin_ws*.

```
$ mkdir -p catkin_ws/src
$ cd catkin_ws
$ catkin_make
```

El comando *catkin_make* es el encargado de compilar el espacio de trabajo en el que vamos a implementar nuestros paquetes.

2. Creación de un paquete ROS

Como ya sabemos, un paquete ROS necesita una serie de ficheros específicos para poder realizar la compilación de dicho paquete. Para ello, en vez de generar nosotros esos ficheros manualmente, vamos a ejecutar el comando *catkin* para que se genere automática toda esa estructura de paquete ROS. Este comando debe ser lanzado desde la carpeta *src/* de nuestro espacio de trabajo.

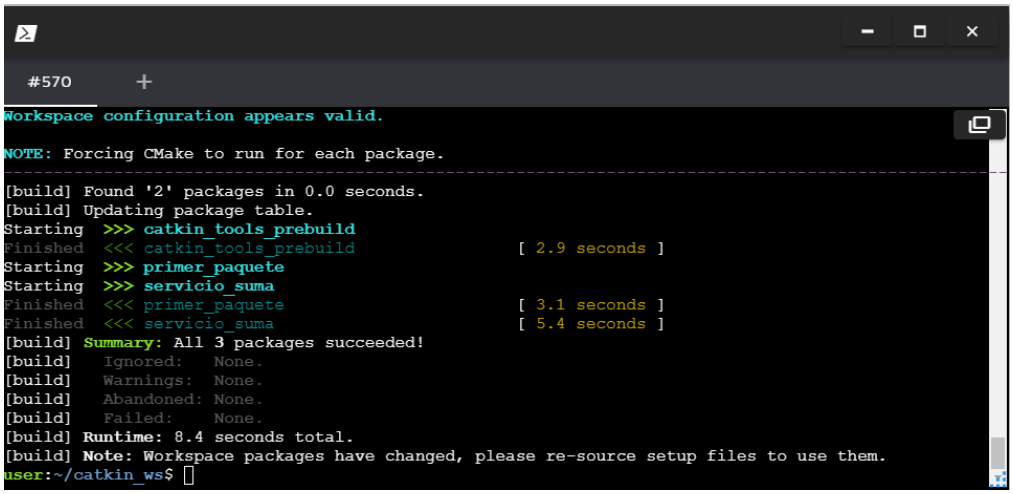
```
$ cd catkin_ws/src
$ catkin_create_pkg primer_paquete rospy std_msgs
```

Inicialmente, como vamos a estar desarrollando código con Python hemos generado este paquete mediante las dependencias *rospy*. Además, hemos añadido a nuestro paquete la dependencia de a los mensajes estándares de ROS, *std_msgs*.

A continuación, vamos a compilar este paquete, aunque nuestro paquete todavía esté vacío, mediante el siguiente comando. Este comando debe ser ejecutado desde el directorio raíz de nuestro espacio de trabajo.

```
$ catkin_make
```

Nos debe aparecer algo parecido a la siguiente imagen. Como vemos nos aparecen los paquetes que se han compilado correctamente y el tiempo que ha tardado en ello, entre otra mucha información. De esta manera, ya podemos empezar a trabajar en nuestro paquete ROS.



P1. ¿Dónde se puede observar dentro de nuestro paquete que realmente tiene como dependencias *rospy* y *std_msgs*?

3. Creando nuestro propio mensaje

En este apartado vamos a ver todo lo necesario para poder crear nuestro propio mensaje y como publicar este tipo de mensaje desde un nodo. Lo primero que tenemos que hacer es crear una carpeta *msg/* y un fichero que se va a llamar *miMensaje.msg*.

```
$ cd catkin_ws/src/primer_paquete
$ mkdir msg
$ cd msg/
$ touch miMensaje.msg
```

Una vez creado el fichero para crear el mensaje, lo editamos para que nuestro mensaje tenga los siguientes valores: *int32 x*, *int32 y*, *string nombre=miNombre*. Tras modificar este fichero, debemos abrir el fichero *package.xml* de nuestro paquete y añadir las siguientes dependencias:

```
<build_depend>message_generation</build_depend>
<exec_depend>message_runtime</exec_depend>
```

A continuación, abriremos el fichero *CMakeLists.txt* de nuestro paquete y modificaremos, sin quitar lo que ya hay o quitando los comentarios correspondientes, las siguientes dependencias:

- *find_package(… REQUIRED COMPONENTS message_generation …).*
- *add_message_files(FILES miMensaje.msg)*
- *generate_messages(DEPENDENCIES std_msgs)*
- *catkin_package(… CATKIN_DEPENS message_runtime …)*

Una vez hecho estas modificaciones, el siguiente paso es volver a compilar el paquete mediante *catkin build primer_paquete* (de esta forma si hay varios paquetes no perderemos tiempo recompilando el resto. Tras compilar correctamente el paquete, debemos actualizar las dependencias de nuestro espacio de trabajo ejecutando *source devel/setup.bash*. Si nuestro mensaje ha sido creado correctamente, podremos saber de qué tipo es el mensaje creado a través de la terminal.

P2. ¿Qué comando debemos utilizar?

Ahora ya estamos a listos para poder crear un nodo que publique en un topic información con el mensaje creado. Para ello, en la carpeta *src/* de nuestro paquete vamos a crear dos nodos, un *publisher*, llamado */envio_miMensaje*, y un *subscriber*, llamado */recibo_miMensaje*.

Para crear estos dos nodos, que vamos a desarrollar en Python, ejecutamos los siguientes comandos.

```
$ cd catkin_ws/src/primer_paquete/src
$ touch nodo_envia.py
$ touch nodo_recibe.py
```

Y escribimos el siguiente código en cada uno de estos ficheros.

nodo_recibe.py

```
#!/usr/bin/env python
import rospy
from primer_paquete.msg import miMensaje

def callback(data):
    rospy.loginfo("Recibo: %s - x: %s; y: %s" % (data.nombre, data.x, data.y))

def nodo_recibe():

    rospy.init_node('nodo_recibe', anonymous=True)

    rospy.Subscriber("mi_topic",  miMensaje, callback)

    rospy.spin()

if __name__ == '__main__':
    nodo_recibe()
```

nodo_envia.py

```
#!/usr/bin/env python
import rospy
from primer_paquete.msg import miMensaje

def nodo_envia():

    mensaje = miMensaje()
    mensaje.x = 0
    mensaje.y = 0

    pub = rospy.Publisher('/mi_topic',  miMensaje, queue_size=10)
    rospy.init_node('nodo_envia', anonymous=True)
    rate = rospy.Rate(10) # 10hz

    while not rospy.is_shutdown():
        mensaje.x += 1
        mensaje.y += 2
        envio_str = "Envio: %s - x: %s; y: %s" % (mensaje.nombre, mensaje.x, mensaje.y)
        rospy.loginfo(envio_str)
        pub.publish(mensaje)
        rate.sleep()

if __name__ == '__main__':

        nodo_envia()
```

Ahora solo nos queda hacer ejecutables estos ficheros, lanzar el ROSMaster (roscore) y lanzar cada nodo en una terminal diferente.

P3. Muéstrame el gráfico de entorno ROS generado, usando rqt_graph.

P4. Intenta modificar la constante *nombre*, del mensaje, modificando el código del nodo */nodo_envia*. ¿Qué sucede?

P5. ¿De qué tipo es el topic que aparece?

4. Creando nuestro propio launch

Como ya hemos visto, somos capaces de lanzar los dos nodos definidos en el punto 3 en terminales separadas y usando nuestro mensaje personalizado. Ahora vamos a aprender como lanzar ambos nodos desde un fichero **.launch*.

Para empezar, debemos crear nuestra carpeta *launch/* y un fichero **.launch*. Para ello ejecutamos los siguientes comandos:

```
$ cd catkin_ws/src/primer_paquete
$ mkdir launch && cd launch
$ touch mi_launch.launch
```

A continuación, introducimos el siguiente código en el fichero *mi_launch.launch*.

```
<launch>
  <!-- Nodo de envio -->
  <node name="envia_nodo" pkg="primer_paquete" type="nodo_envia.py" output="screen"/>

  <!-- Nodo de recibir -->
  <node name="recibe_nodo" pkg="primer_paquete" type="nodo_recibe.py" output="screen"/>
</launch>
```

P6. ¿Con que nombre se han lanzado los nodos?

Ejercicios

1. Crea un paquete nuevo que se llame p2*pkg*. A continuación, define un mensaje con los siguientes campos:
   - *int32 numero*
   - *geometry_msgs/Pose posicion*
   - *string fecha=hoy*

   Una vez definido, muestra los campos del mensaje usando el comando *rosmsg show*.

2. Crea un nodo *publisher*, llamado */nodopub_ejercicio2*, y un nodo *subscriber*, llamado */nodosub_ejercicio2*, que sean capaces de enviar y recibir el tipo de mensajes a través un topic al que llamaremos */topic_ejercicio2*. Estos nodos deben ser creados dentro del paquete ROS p2*pkg*.

   El nodo */nodopub_ejercicio2* debe tener un argumento de entrada que se asignará al campo *numero* del mensaje. Además, para enviar todos los valores del campo *posición* utiliza la función *random* del paquete *random* de Python.

Ambos nodos deben mostrar por la terminal el valor del campo *fecha*, del campo *numero*, del campo *posicion.position.x* y del campo *posicion.orientation.w*

(
pistas:
from random import random
posicion.orientation.x = random()
)

3. Genera un fichero llamado *launch_ejercicio3.launch* que sea capaz de lanzar los nodos creados en el ejercicio anterior al mismo tiempo. El argumento de entrada se debe añadir desde el propio fichero *launch*, teniendo un valor por defecto de *3*.

4. Crea otro fichero llamado *launch_ejercicio4.launch* que agrupe estos dos nodos dentro de un mismo grupo, llamado *miGrupo*. Haz uso de la etiqueta *remap* para que el *topic* también se encuentre dentro de ese grupo.

    - Comprueba que tanto los nodos como los topicos creados estan dentro del espacio de trabajo miGrupo.
    - Muestra las capturas de pantalla.

Entrega

1. Tienes que entregar un fichero pdf con las contestaciones a las cuestiones planteadas y tambien con las capturas que demuestren el funcionamiento de los ejercicios.
2. Tambien un fichero con el paquete creado.
3. Todo esto comprimido en un unico fichero llamado P2.zip