

Práctica 2

Introducción a la Programación Robótica

Creación de un publicador-suscriptor

Enunciado

Partiendo de lo visto a través del ejemplo de operativa en el modelo publicador-suscriptor disponible en la página web de ROS, en esta segunda práctica se propone la realización de un par de scripts sencillos que permitan capturar y manejar datos de memoria del sistema operativo.

Por tanto, el **objetivo** de la práctica es crear un publicador que haga disponible el valor en megabytes de memoria RAM total, disponible y libre, así como el valor en kilobytes de los buffers de memoria del sistema, a una frecuencia de 1 Hz.

El desarrollo de la práctica se dividirá en 4 partes, en las que comenzaremos con una versión simplificada de los scripts publicador-suscriptor, que completaremos en un segundo momento:

- 1) [creación de una primera versión del publicador](#)
- 2) [creación de una primera versión del suscriptor](#)
- 3) [ampliación de la lógica del publicador](#)
- 4) [ampliación de la lógica del suscriptor](#)

Pasos a realizar:

1) creación de una primera versión del publicador

Objetivo: el publicador tomará únicamente el dato de memoria libre y lo hará disponible.

Realiza los siguientes pasos:

- a) crea un paquete nuevo al que llamaremos **simple_pub_sub** con las dependencias **std_msgs**, **rospy**, **roscpp**.

```
# cd /home/robot/catkin_ws/src
```

```
# catkin_create_pkg simple_pub_sub std_msgs rospy roscpp
```

- b) crea la carpeta **scripts**.
- c) crea dentro el fichero **py_publisher.py** con el siguiente código:

```
#!/usr/bin/env python

# Importamos los módulos necesarios
import rospy
from std_msgs.msg import Float64
import re

def memory_publisher():
    # Asigna un nombre al nodo
    rospy.init_node('memory_publisher', anonymous=True)
    # Asigna un nombre al nodo, tipo de mensaje y tamaño de la cola
    pub = rospy.Publisher('memory', Float64, queue_size=1000)
    # 1 Hz
    rate = rospy.Rate(1)
    msg = Float64(0.0)
    # Devuelve true si pulsamos CTRL+C
    while not rospy.is_shutdown():
        get_free_memory(msg)
        rospy.loginfo("Sending free memory value: %f", msg.data)
        pub.publish(msg)
    # Detenemos la ejecución lo justo para iterar a 1 Hz
    rate.sleep()

def get_free_memory(msg):
    # Abre el fichero /proc/meminfo
    with open('/proc/meminfo') as fp:
        # Lee la primera línea, cuyo valor (memoria total) no guardaremos
        fp.readline()
        # Lee la segunda línea que contiene el valor de memoria libre
        line = fp.readline()
        # Divide la línea para quedarse con el valor
        str_list = re.split('[\t| ]+', line)
        # Asigna a msg.data el valor de memoria en megabytes.
        msg.data = long(str_list[1]) / 1024.0

if __name__ == '__main__':
    try:
        memory_publisher()
    except rospy.ROSInterruptException:
        pass
```

Algunas observaciones sobre el código:

- La variable `__name__` toma el valor `__main__` al ser iniciado.
- En la función **memory_publisher**:
 - **anonymous=True**, asegura que el nodo obtiene un nombre único añadiendo un número aleatorio al final del nombre. Si dos nodos responden a un mismo nombre, el primero es descartado.
 - **rospy.is_shutdown()** devuelve true si presionamos CTRL+C.
 - **rospy.ROSInterruptException** puede lanzarse cuando se ejecuta `sleep()` y CTRL+C es presionado o si el nodo se para de otra forma.

- d) completa ahora el código de la función **get_memory_free**. Esta función tiene por objetivo averiguar la cantidad de memoria libre que tiene el sistema en cierto momento. Una forma de hacerlo es obtener el dato del fichero virtual **/proc/meminfo**.
- e) otorga al fichero permisos de ejecución.
- f) ejecuta el nodo **roswin** (asegúrate previamente que roscore está activo).

2) creación de una primera versión del suscriptor

Objetivo: en este caso crearemos un suscriptor que sea capaz de recoger los datos publicados por el publicador de memoria.

Realiza los siguientes pasos:

- g) crea dentro de la carpeta **scripts** el fichero **py_subscriber.py** y copia el siguiente código en él:

```
#!/usr/bin/env python

import rospy
from std_msgs.msg import Float64

# La función "callback" se ejecuta cuando un mensaje Float64 llega.
def callback(msg):
```

```
rospy.loginfo("Received free memory: %f", msg.data)

def memory_subscriber():
    rospy.init_node('memory_subscriber', anonymous=True)
    # El nombre de tópicos debe corresponder al del publicador, al igual que el tipo de
    # mensaje
    rospy.Subscriber("memory", Float64, callback)
    # spin() mantiene la ejecución hasta que el nodo es cancelado mediante CTRL+C
    rospy.spin()
if __name__ == '__main__':
    memory_subscriber()
```

- Al crear el suscriptor, el nombre del tópico y tipo del mensaje deben coincidir con los del publicador.
- El suscriptor recibe además una función *callback* que se ejecuta cada vez que llega un mensaje por el tópico.
- La función *callback* se ejecuta en un hilo propio.
- La función *spin* bloquea el hilo principal hasta que se presiona CTRL+C.

h) Dale permisos de ejecución.

i) Ejecuta el nodo con **roslaunch** junto con el nodo publicador y observa la comunicación entre ambos.

3) ampliación de la lógica del publicador

Objetivo: crear un publicador que publique:

- el valor en megabytes de memoria RAM total
- el valor en megabytes de memoria disponible
- el valor en megabytes de memoria libre
- el valor en kilobytes de los buffers de memoria del sistema, a una frecuencia de 1 Hz.

Para ello, vamos a leer de nuevo el contenido del fichero **/proc/meminfo** y crear nuestro propio mensaje de ROS. Necesitamos tomar 4 valores del fichero, que aparecen en las cuatro primeras líneas del mismo.

Acto seguido, publicaremos dichos valores juntos en un mismo mensaje de ROS que crearemos desde cero.

> Creación del paquete *custom_pub_sub*

Realiza los siguientes pasos:

- a) Crea un **paquete** nuevo **custom_pub_sub** con las dependencias **std_msgs**, **rospy**, **roscpp**.
- b) Crea la carpeta **msg**.
- c) Crea dentro el fichero **Memory.msg** y copia el siguiente código:

```
# El encabezado contiene información de uso común en mensajes ROS
# Se usa normalmente como primer campo de los mensajes
Header header
std_msgs/Float64 total #Megabytes
std_msgs/Float64 available #Megabytes
std_msgs/Float64 free #Megabytes
# También se puede usar datos primitivos
uint32 buffers #Kilobytes
```

Observaciones:

El mensaje está compuesto por cinco campos, que se referirán a los diferentes datos que vamos a guardar:

- Un mensaje tipo **Header (std_msgs/Header)**
 - Normalmente es de uso no obligatorio, aunque suele utilizarse como primer campo de mensajes nuevos.
 - Si usamos determinadas librerías (como tf) su uso sí es obligatorio.
 - En la definición puede aparecer con su nombre completo (std_msgs/Header).
 - Se utiliza en tipos de datos que requieren una marca de tiempo.
 - Está compuesto de tres campos: seq, stamp y frame.
- Tres mensajes **Float64**
- Un tipo primitivo **uint32**.

- d) A continuación debemos incluir la referencia al nuevo mensaje en el fichero

CMakeLists.txt. Sustituye su contenido por el siguiente contenido:

```
cmake_minimum_required(VERSION 2.8.3)
project(custom_pub_sub)

## Support for c++11
add_compile_options(-std=c++11)

## Find catkin macros and libraries
find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  std_msgs
  message_generation
)

## Generate messages in the 'msg' folder
add_message_files(
  FILES
  Memory.msg
)

## Generate added messages and services with any dependencies listed here
generate_messages(
  DEPENDENCIES
  std_msgs
)

## The catkin_package macro generates cmake config files for the package
catkin_package()

include_directories(
  # include
  ${catkin_INCLUDE_DIRS}
)

#####
### PUBLISHER ###
#####

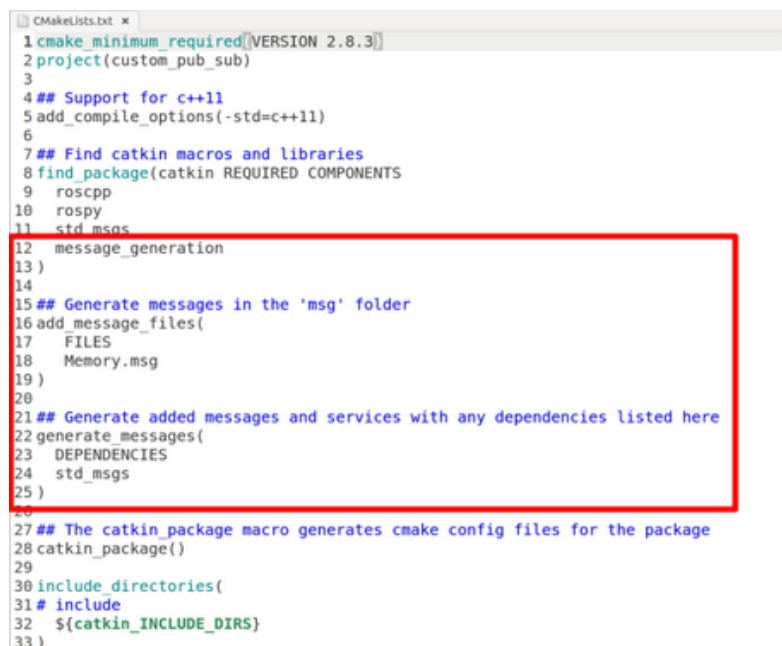
## Declare a C++ executable
#add_executable(${PROJECT_NAME}_publisher_node src/publisher.cpp)
## Rename C++ executable without prefix
#set_target_properties(${PROJECT_NAME}_publisher_node PROPERTIES OUTPUT_NAME publisher_node PREFIX "")
## Add cmake target dependencies of the executable
#add_dependencies(${PROJECT_NAME}_publisher_node ${${PROJECT_NAME}_EXPORTED_TARGETS}
```

```
{catkin_EXPORTED_TARGETS})
## Specify libraries to link a library or executable target against
#target_link_libraries(${PROJECT_NAME}_publisher_node
# ${catkin_LIBRARIES}
#)

#####
### SUBSCRIBER ###
#####

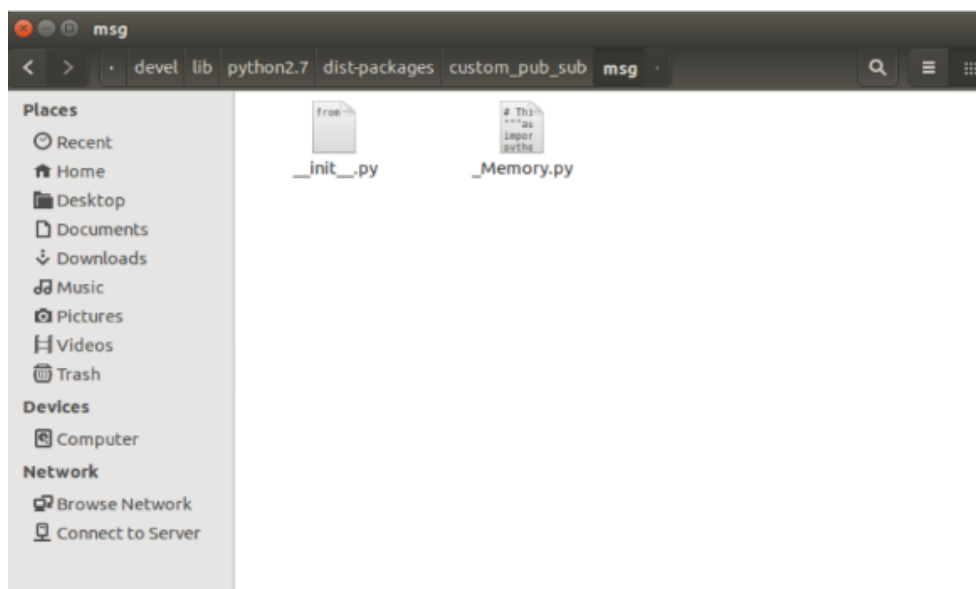
## Declare a C++ executable
#add_executable(${PROJECT_NAME}_subscriber_node src/subscriber.cpp)
## Rename C++ executable without prefix
#set_target_properties(${PROJECT_NAME}_subscriber_node PROPERTIES OUTPUT_NAME subscriber_node PREFIX
"")
## Add cmake target dependencies of the executable
#add_dependencies(${PROJECT_NAME}_subscriber_node ${${PROJECT_NAME}_EXPORTED_TARGETS}
${catkin_EXPORTED_TARGETS})
## Specify libraries to link a library or executable target against
#target_link_libraries(${PROJECT_NAME}_subscriber_node
# ${catkin_LIBRARIES}
#)
```

Asegúrate que la siguiente sección queda como se muestra en esta imagen:



```
CMakeLists.txt
1 cmake_minimum_required(VERSION 2.8.3)
2 project(custom_pub_sub)
3
4 ## Support for c++11
5 add_compile_options(-std=c++11)
6
7 ## Find catkin macros and libraries
8 find_package(catkin REQUIRED COMPONENTS
9   roscpp
10  rospy
11  std_msgs
12  message_generation
13 )
14
15 ## Generate messages in the 'msg' folder
16 add_message_files(
17   FILES
18   Memory.msg
19 )
20
21 ## Generate added messages and services with any dependencies listed here
22 generate_messages(
23   DEPENDENCIES
24   std_msgs
25 )
26
27 ## The catkin package macro generates cmake config files for the package
28 catkin_package()
29
30 include_directories(
31   # include
32   ${catkin_INCLUDE_DIRS}
33 )
```

- e) Compila con [catkin make](#).
- f) Comprueba el código autogenerated (ejemplo del caso Python)



NOTA: A pesar de que Python es un lenguaje interpretado y que en principio no es necesario tocar el fichero **CMakeLists.txt**, ni compilarlo, en esta ocasión sí lo es puesto que necesitamos generar el código de nuestro nuevo mensaje. Por tanto, **hay que compilar con catkin_make obligatoriamente cada vez que haya que generar mensajes.**

Una vez generado el paquete, ahora pasaremos a la codificación del script en Python.

> Creación del script **py_publisher.py**

Realiza los siguientes pasos:

- a) Crea la carpeta **scripts**.
- b) Crea dentro el fichero **py_publisher.py** y copia el siguiente código. Completa las líneas de la función **get_memory** a partir de las sugerencias que tienes en los comentarios:

```
#!/usr/bin/env python
```



```
import rospy
from custom_pub_sub.msg import Memory
import re

def talker():
    rospy.init_node('memory_publisher', anonymous=True)
    pub = rospy.Publisher('memory', Memory, queue_size=1000)
    rate = rospy.Rate(1) # 1 Hz
    counter = 0
    msg = Memory()
    while not rospy.is_shutdown():
        msg.header.seq = counter
        msg.header.stamp = rospy.Time.now()
        msg.header.frame_id = "0"
        get_memory(msg)
        rospy.loginfo("Sending memory message of time: %d.%d", msg.header.stamp.secs,
msg.header.stamp.nsecs)
        pub.publish(msg)
        rate.sleep()
        counter += 1

def get_memory(msg):
    # Abre el fichero /proc/meminfo'
    with open('/proc/meminfo') as fp:
        # Lee la primera línea con el valor de memoria total, guardando el valor
        # en 'msg.total.data'
        line = fp.readline()
        msg.total.data = get_number(line) / 1024.0
        # Lee la segunda línea que contiene el valor de memoria libre, guardándolo el valor
        # en megabytes en 'msg.free.data'
        line = fp.readline()
        msg.free.data = get_number(line) / 1024.0
        # Lee la tercera línea que contiene el valor de memoria libre, guardándolo
        # en megabytes en 'msg.available.data'
        line = fp.readline()
        msg.available.data = get_number(line) / 1024.0
        # Lee la cuarta línea que contiene el valor de memoria buffer, guardándolo en
        # megabytes en 'msg.buffers'
        line = fp.readline()
        msg.buffers = get_number(line)

def get_number(line):
    str_list = re.split('[\t] +', line)
    return long(str_list[1])

if __name__ == '__main__':
    try:
        talker()
```

```
except rospy.ROSInterruptException:  
    pass
```

Observaciones:

- El nuevo mensaje se importa e se instancia de la misma forma que un mensaje estándar.
- Rellenamos el campo de cabecera con un contador, hora actual y *string* por defecto.
- Se ha creado una función de ayuda **get_number** para extraer el campo numérico de una línea del
- fichero de texto.
- Imprime por pantalla la hora del mensaje.

c) Otorga permisos de ejecución.

d) Ejecuta el nodo con **roslun**.

4) ampliación de la lógica del suscriptor

Objetivo: Crear un suscriptor que sea capaz de recoger los datos publicados por el publicador de memoria.

Realiza los siguientes pasos:

- a) Crea dentro de la carpeta **scripts** el fichero **py_subscriber.py** y copia el siguiente código:

```
#!/usr/bin/env python  
  
import rospy  
from custom_pub_sub.msg import Memory  
  
def callback(msg):  
    rospy.loginfo("Received memory message of time: %d.%d", msg.header.stamp.secs,  
msg.header.stamp.nsecs)  
    print 'Total memory: ' + str(msg.total.data)  
    print 'Free memory: ' + str(msg.free.data)  
    print 'Available memory: ' + str(msg.available.data)  
    print 'Buffers memory: ' + str(msg.buffers)  
    print "
```

```
def listener():  
    rospy.init_node('memory_subscriber', anonymous=True)  
    rospy.Subscriber("memory", Memory, callback)  
    # spin() simply keeps python from exiting until this node is stopped  
    rospy.spin()  
  
if __name__ == '__main__':  
    listener()
```

- b) Otórgale permisos de ejecución.
- c) Ejecutar el nodo con **roslaunch** junto con el nodo publicador y observar la comunicación entre ambos.