

Sensores y Actuadores

Clase 9

Sensores



Crear paquete “sensactu”

```
alex@alex-VirtualBox:~/catkin_ws/src$ cd catkin_ws/src/
alex@alex-VirtualBox:~/catkin_ws/src$ catkin_create_pkg sensactio rospym
Created file sensactio/CMakeLists.txt
Created file sensactio/package.xml
Created folder sensactio/src
Successfully created files in /home/alex/catkin_ws/src/sensactio. Please adjust the
values in package.xml.
alex@alex-VirtualBox:~/catkin_ws/src$
```

```
alex@alex-VirtualBox:~/catkin_ws$ catkin_make
base path: /home/alex/catkin_ws
source space: /home/alex/catkin_ws/src
build path: /home/alex/catkin_ws/build
devel space: /home/alex/catkin_ws/devel
install space: /home/alex/catkin_ws/install

# make /home/alex/catkin_ws/src -DCATKIN_DEVEL_PREFIX=/home/alex/catkin_ws/devel -DCMAKE_INSTALL_PREFIX=/home/alex/catkin_ws/install -G Unix Makefiles" in "/home/alex/catkin_ws/build"
```

Compilación del paquete

```
alex@alex-virtualbox: ~/catkin_ws
-- new_package
-- sensactl

+++ processing catkin package: 'new_package'
-- add_subdirectory(new_package)
Using these message generators: gencpp;genlisp;genpy
new_package: 0 messages, 1 services
+++ processing catkin package: 'sensactl'
-- add_subdirectory(sensactl)
Configuring done
Generating done
Build files have been written to: /home/alex/catkin_ws/build

alex@alex-virtualbox: ~/catkin_ws
alex@alex-virtualbox:~/catkin_ws$ make -j1 -l1 -C /home/alex/catkin_ws/build
make[1]: Entering directory '/home/alex/catkin_ws/build'
[ 0%] Built target new_package_generate_messages_check_deps_WordCount
[ 0%] Built target std_msgs_generate_messages_lisp
[ 25%] Built target new_package_generate_messages_lisp
[ 25%] Built target std_msgs_generate_messages_py
[ 75%] Built target new_package_generate_messages_py
[ 75%] Built target std_msgs_generate_messages_cpp
[100%] Built target new_package_generate_messages_cpp
[100%] Built target new_package_generate_messages
alex@alex-virtualbox:~/catkin_ws$
```

Habilitar la librería FakeSensor

- Descargamos el archivo “fake_sensor.py”.
- Se copia el archivo en la dirección indicada en la figura.
- Se da los permisos necesarios para trabajar con el archivo.
- Luego se importará algunos sensores o actuadores.

```
alex@alex-VirtualBox: ~/catkin_ws/src/sensactu/src
alex@alex-VirtualBox:~/catkin_ws/src/sensactu/src$ chmod u+x fake_sensor.py
```

Obtener las medidas del sensor

```
#!/usr/bin/env python
from math import pi
from fake_sensor import FakeSensor
import rospy
import tf
from geometry_msgs.msg import Quaternion

def make_quaternion(angle):
    q = tf.transformations.quaternion_from_euler(0, 0, angle)
    return Quaternion(*q)

if __name__ == '__main__':
    sensor = FakeSensor()
    rospy.init_node('fake_sensor')
    pub = rospy.Publisher('angle', Quaternion, queue_size=10)
    rate = rospy.Rate(10.0)
    while not rospy.is_shutdown():
        angle = sensor.sensor.value() * 2 * pi / 100.0
        q = make_quaternion(angle)
        pub.publish(q)
        rate.sleep()
```

Se importa la librería FakeSensor

Tipo de dato para ubicación de un móvil

Cambio de tipo de dato, de ángulos a Quaternion

Variable tipo FakeSensor

Obtener las medidas del sensor

```
#!/usr/bin/env python
from math import pi
from fake_sensor import FakeSensor
import rospy
import tf
from geometry_msgs.msg import Quaternion

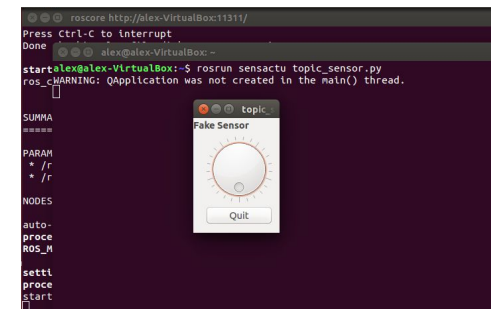
def make_quaternion(angle):
    q = tf.transformations.quaternion_from_euler(0, 0, angle)
    return Quaternion(*q)

if __name__ == '__main__':
    sensor = FakeSensor()
    rospy.init_node('fake_sensor')
    pub = rospy.Publisher('angle', Quaternion, queue_size=10)
    rate = rospy.Rate(10.0)
    while not rospy.is_shutdown():
        angle = sensor.sensor.value() * 2 * pi / 100.0
        q = make_quaternion(angle)
        pub.publish(q)
        rate.sleep()
```

Se muestra recibe el valor del sensor cada 10 hz

Se hace la transformación del ángulo

Obtener las medidas del sensor



Lista de los tópicos

```
roscore http://alex-VirtualBox:11311/
Press Ctrl-C to interrupt
Done
alex@alex-VirtualBox:~$ rosrun sensactu topic_sensor.py
ros_cWARNING: QApplication was not created in the main() thread.

alex@alex-VirtualBox:~$ rostopic list
SUMMA
=====
/angle
/rosout
PARAM
* /r
* /r
NODES
auto-
proce
ROS_M
setti
proce
start
Quit
```

Transmisión de data

```
roscore http://alex-VirtualBox:11311/
Press Ctrl-C to interrupt
Done
alex@alex-VirtualBox:~$ rosrun sensactu topic_sensor.py
ros_cWARNING: QApplication was not created in the main() thread.

alex@alex-VirtualBox:~$ rostopic hz angle
SUMMA
=====
subscribed to [/angle]
PARAM
* /r
* /r
average rate: 9.997
min: 0.100s max: 0.101s std dev: 0.00034s window: 10
average rate: 9.998
min: 0.099s max: 0.101s std dev: 0.00032s window: 20
average rate: 9.998
min: 0.099s max: 0.101s std dev: 0.00049s window: 30
average rate: 9.999
min: 0.099s max: 0.101s std dev: 0.00044s window: 40
^Coverage rate: 9.999
min: 0.099s max: 0.101s std dev: 0.00043s window: 47
alex@alex-VirtualBox:~$
```

Mostrar la data del sensor

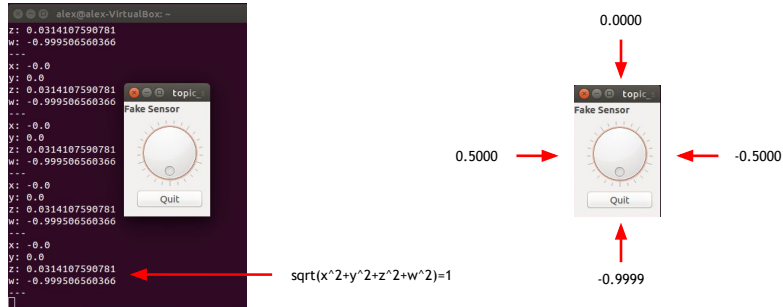
```
roscore http://alex-VirtualBox:11311/
Press Ctrl-C to interrupt
Done
alex@alex-VirtualBox:~$ rosrun sensactu topic_sensor.py
ros_cWARNING: QApplication was not created in the main() thread.

alex@alex-VirtualBox:~$ rostopic echo -n 1 angle
SUMMA
=====
x: 0.0
y: 0.0
PARAM
* /r
* /r
w: 1.0
NODES
auto-
proce
ROS_M
setti
proce
start
Quit
```

Mostrar la data del sensor

```
alex@alex-VirtualBox:~$ rostopic echo -n 1 angle
SUMMA
=====
x: 0.0
y: 0.0
PARAM
* /r
* /r
w: 1.0
NODES
auto-
proce
ROS_M
setti
proce
start
Quit
```

Mostrar la data del sensor

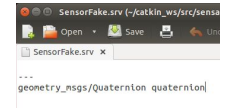


Servicio para un sensor

```
#!/usr/bin/env python
from math import pi
from fake_sensor import FakeSensor

import rospy
import tf
from geometry_msgs.msg import Quaternion
from sensactu.srv import SensorFake, SensorFakeResponse

def make_quaternion(angle):
    q = tf.transformations.quaternion_from_euler(0, 0, angle)
    return Quaternion(*q)
```



Servicio para un sensor

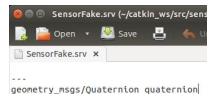
```
def callback(request):
    global sensor
    angle = sensor.sensor.value * 2 * pi / 100.0
    q = make_quaternion(angle)

    return SensorFakeResponse(q)

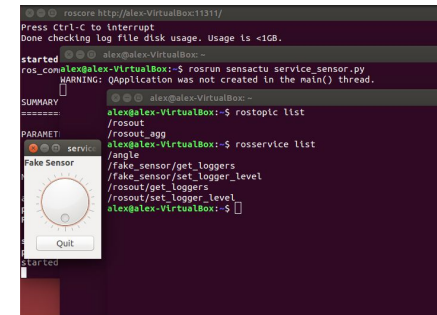
if __name__ == '__main__':
    global sensor
    sensor = FakeSensor()

    rospy.init_node('fake_sensor')
    service = rospy.Service('angle', SensorFake, callback)

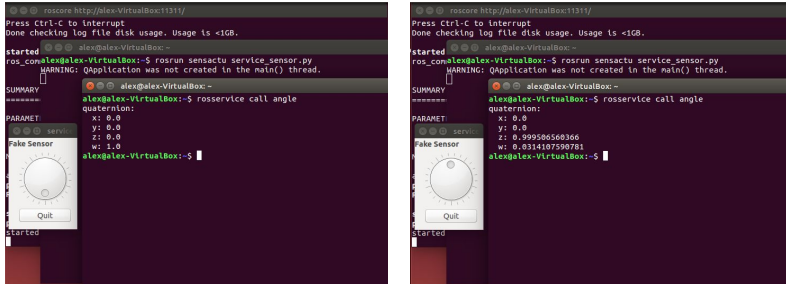
    rospy.spin()
```



Servicio y tópicos del sensor

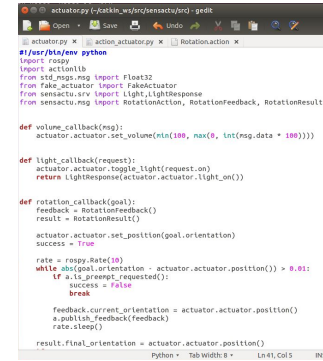


Servicio y tópicos del sensor



The image shows two terminal windows side-by-side. The left terminal is running a ROS node named 'sensactu' and has a 'Fake Sensor' GUI window open. The right terminal is running a ROS node named 'rosservice' and has a 'Fake Sensor' GUI window open. Both terminals show the output of the 'rosservice call angle' command, which returns a quaternion: 'x: 0.0, y: 0.0, z: 0.99990560366, w: 0.0314107590781'.

Actuadores



The image shows a ROS node editor window for a node named 'actuator'. The code is written in Python and defines a 'FakeActuator' class. It includes a 'volume_callback' method that sets the volume of the actuator based on a message. It also includes a 'light_callback' method that toggles the light of the actuator. The code is as follows:

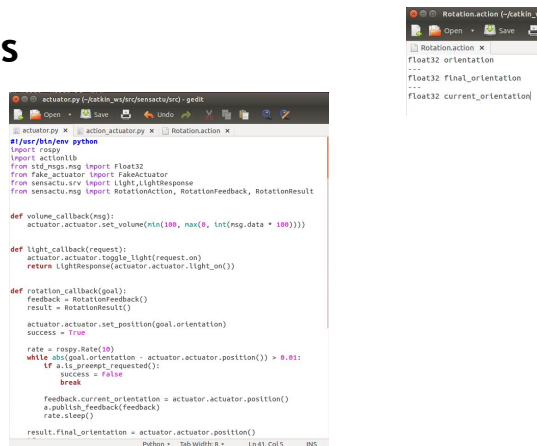
```
#!/usr/bin/env python
import rospy
import actionlib
from std_msgs.msg import Float32
from fake_actuator import FakeActuator
from sensactu.srv import Light, LightResponse
from sensactu.msg import RotationAction, RotationFeedback, RotationResult

def volume_callback(msg):
    actuator.actuator.set_volume(min(100, max(0, int(msg.data * 100))))

def light_callback(request):
    actuator.actuator.toggle_light(request.on)
    return LightResponse(actuator.actuator.light_on())

def rotation_callback(goal):
    feedback = RotationFeedback()
    result = RotationResult()
    actuator.actuator.set_position(goal.orientation)
    success = True
    rate = rospy.Rate(10)
    while abs(goal.orientation - actuator.actuator.position()) > 0.01:
        if a.is_preempt_requested():
            success = False
            break
        feedback.current_orientation = actuator.actuator.position()
        a.publish_feedback(feedback)
        rate.sleep()
    result.final_orientation = actuator.actuator.position()
    return result
```

Actuadores



The image shows a ROS node editor window for a node named 'actuator'. The code is written in Python and defines a 'FakeActuator' class. It includes a 'volume_callback' method that sets the volume of the actuator based on a message. It also includes a 'light_callback' method that toggles the light of the actuator. The code is as follows:

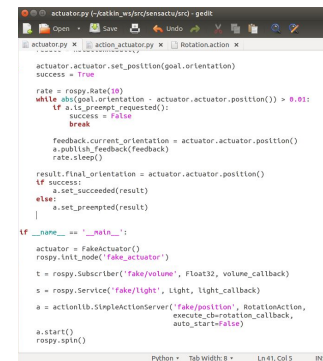
```
#!/usr/bin/env python
import rospy
import actionlib
from std_msgs.msg import Float32
from fake_actuator import FakeActuator
from sensactu.srv import Light, LightResponse
from sensactu.msg import RotationAction, RotationFeedback, RotationResult

def volume_callback(msg):
    actuator.actuator.set_volume(min(100, max(0, int(msg.data * 100))))

def light_callback(request):
    actuator.actuator.toggle_light(request.on)
    return LightResponse(actuator.actuator.light_on())

def rotation_callback(goal):
    feedback = RotationFeedback()
    result = RotationResult()
    actuator.actuator.set_position(goal.orientation)
    success = True
    rate = rospy.Rate(10)
    while abs(goal.orientation - actuator.actuator.position()) > 0.01:
        if a.is_preempt_requested():
            success = False
            break
        feedback.current_orientation = actuator.actuator.position()
        a.publish_feedback(feedback)
        rate.sleep()
    result.final_orientation = actuator.actuator.position()
    return result
```

Actuadores



The image shows a ROS node editor window for a node named 'actuator'. The code is written in Python and defines a 'FakeActuator' class. It includes a 'volume_callback' method that sets the volume of the actuator based on a message. It also includes a 'light_callback' method that toggles the light of the actuator. The code is as follows:

```
#!/usr/bin/env python
import rospy
import actionlib
from std_msgs.msg import Float32
from fake_actuator import FakeActuator
from sensactu.srv import Light, LightResponse
from sensactu.msg import RotationAction, RotationFeedback, RotationResult

def volume_callback(msg):
    actuator.actuator.set_volume(min(100, max(0, int(msg.data * 100))))

def light_callback(request):
    actuator.actuator.toggle_light(request.on)
    return LightResponse(actuator.actuator.light_on())

def rotation_callback(goal):
    feedback = RotationFeedback()
    result = RotationResult()
    actuator.actuator.set_position(goal.orientation)
    success = True
    rate = rospy.Rate(10)
    while abs(goal.orientation - actuator.actuator.position()) > 0.01:
        if a.is_preempt_requested():
            success = False
            break
        feedback.current_orientation = actuator.actuator.position()
        a.publish_feedback(feedback)
        rate.sleep()
    result.final_orientation = actuator.actuator.position()
    return result
```

Implementar configuraciones

```
package.xml X
<?xml version="1.0"?>
<package>
  <name>sensactu</name>
  <version>0.0.1</version>
  <description>Virtual Sensor/Actuators package</description>
  <maintainer email="lopez.alexander@upc.pe">Alexander Lopez</maintainer>
  <license>BSD</license>
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_depend>message_generation</build_depend>
  <build_depend>actionlib_msgs</build_depend>
  <run_depend>actionlib_msgs</run_depend>
  <run_depend>message_runtime</run_depend>
  <run_depend>std_msgs</run_depend>
  <run_depend>roscpp</run_depend>

  <!-- The export tag contains other, unspecified, tags -->
  <export>
    <!-- Other tools can request additional information be placed here -->
  </export>
</package>
```

Implementar configuraciones

```
CMakeLists.txt X
cmake_minimum_required(VERSION 2.8.3)
project(sensactu)

## Add support for C++11, supported in ROS Kinetic and newer
# add_definitions(-std=c++11)

## Find catkin macros and libraries
## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
## is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS
  roscpp
  std_msgs
  message_generation
  geometry_msgs
  actionlib_msgs
)

## System dependencies are found with CMake's conventions
# find_package(Boost REQUIRED COMPONENTS system)

## Uncomment this if the package has a setup.py. This macro ensures
## modules and global scripts declared therein get installed
## See http://ros.org/doc/api/catkin/html/user_guide/setup_dot_py.html
# catkin_python_setup()
```

Implementar configuraciones

```
CMakeLists.txt X
#
## Generate services in the 'srv' folder
add_service_files(
  FILES
  SensorFake.srv
  Light.srv
  Service1.srv
  Service2.srv
)

## Generate actions in the 'action' folder
add_action_files(
  DIRECTORY action
  FILES Rotation.action
  Action1.action
  Action2.action
)

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

Implementar configuraciones

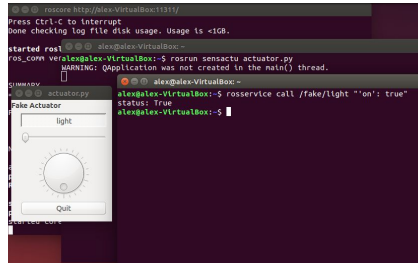
```
CMakeLists.txt X
#
## CATKIN_DEPENDS: catkin_packages dependent projects also need
## DEPENDS: system dependencies of this project that dependent projects
## also need
catkin_package(
  # INCLUDE_DIRS include
  # LIBRARIES sensactu
  CATKIN_DEPENDS message_runtime roscpp std_msgs actionlib_msgs
  DEPENDS system_lib
)

#####
## Build ##
#####

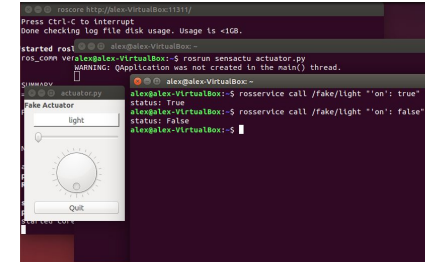
## Specify additional locations of header files
## Your package locations should be listed before other locations
include_directories(include)
include_directories(
  ${catkin_INCLUDE_DIRS}
)

## Declare a C++ library
# add_library(${PROJECT_NAME})
# src/${PROJECT_NAME}/sensactu.cpp
# )
```

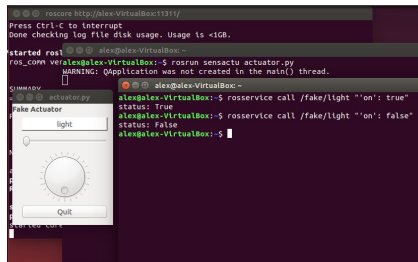
Publicación



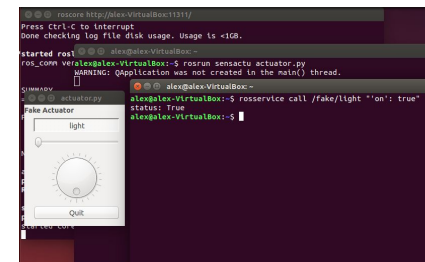
Publicación



Servicio



Servicio



Implementación de una Acción

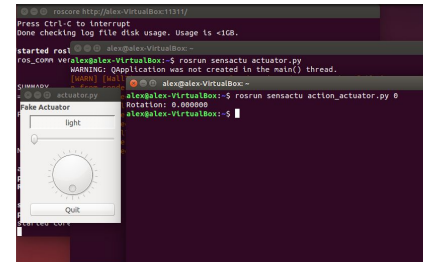
```
action_actuator.py
#!/usr/bin/env python
import sys
import rospy
import actionlib
from sensactu.msg import RotationAction, RotationGoal, RotationResult,
RotationFeedback

rospy.init_node('action_actuator')
client = actionlib.SimpleActionClient('fake/position', RotationAction)
client.wait_for_server()

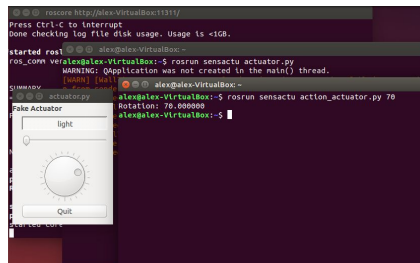
goal = RotationGoal()
goal.orientation = int(' '.join(sys.argv[1:]))
client.send_goal(goal)
client.wait_for_result()
print('Rotation: %f' % (client.get_result().final_orientation))
```

```
RotationAction
float32 orientation
---
float32 final_orientation
---
float32 current_orientation
```

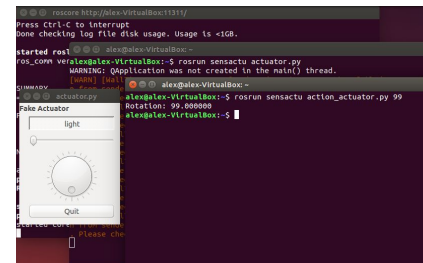
Acción



Acción



Acción



¡Gracias!

¡La única pregunta tonta es la que no se
hace!